UNIVERSIDAD POLITÉCTNICA DE MADRID

Escuela universitaria de Ingeniería
Técnica de Telecomunicación

PROYECTO FIN DE CARRERA

SISTEMA PARA CAMBIAR LA GEOMETRÍA DE UN MOTOR

Jaime Antonio García Padilla

Septiembre 2013

E.U.I.T. TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA
PLAN 2000

TEMA: Fórmula SAE

TÍTULO: Sistema para cambiar la geometría de un motor

AUTOR: Jaime Ant. García Padilla

TUTOR: Pedro Cobos Arribas **Vº Bº.**

DEPARTAMENTO: SEC

Miembros del Tribunal Calificador:

PRESIDENTE: Manuel Vázquez López

VOCAL:

VOCAL SECRETARIO: Franciso Javier Jiménez Martínez

DIRECTOR:

Fecha de lectura: Septiembre 2013

Calificación: **El Secretario,**

RESUMEN DEL PROYECTO:

En este proyecto se diseña y fabrica un sistema para cambiar la geometría de un motor en un monoplaza que compite en la Fórmula SAE.
La variación de la geometría de un motor es un medio que permite un mayor aprovechamiento del par motor, en este caso mejorando la potencia del motor en el rango de revoluciones entre 8000 y 15000 rpm. El método empleado es la variación de la longitud de los tubos de admisión de aire hacia la cámara de combustión.
Esta variación de la distancia de los tubos de admisión se realiza mediante el control de un motor lineal paso a paso, que se mueve según un perfil trapezoidal, que se repite en tramos de 1 mm para conseguir el control de la velocidad media del desplazamiento y mejorando el empuje del motor paso a paso. El control del motor se realiza empleando un microcontrolador, en concreto el PIC32MX795F512-L.
El sistema se activa cuando detecta que se demanda una aceleración y se ha sobrepasado un determinado punto de revoluciones del motor de monoplaza previamente seleccionado. Entonces se actúa sobre el motor lineal paso a paso y con su acción se variará la geometría del motor de combustión.
El sistema se comunica por bus CAN, protocolo de comunicaciones muy extendido en el ámbito de la automoción. El cual se emplea para transmitir información del sistema y recibir órdenes de otros módulos conectados al bus CAN.

AGRADECIMIENTOS.

Primero a mi padre y a mi madre, que ya desde pequeño querían más que yo que fuera a la universidad y yo me quejaba ante la idea de tanto estudio. Por animarme a matricularme ya que he disfrutado mucho de este periodo, gracias papá por tu apoyo y las oportunidades que siempre me has dado para llegar hasta aquí. Sin ninguna duda eres el que más ha confiado en mí. Este trabajo es tanto tuyo como mío.

A mi hermana que me ha escuchado cuando necesitaba contar algo, aunque a veces no me entendiese, siempre ha estado ahí dándome confianza y riéndose conmigo.

Lola, a ti, por tu preocupación interés y apoyo durante todo el tiempo.

Y a mis amigetes y compañeros, ya que sin ellos es muy difícil alcanzar los retos que uno se propone, vosotros que habéis estado conmigo pasando los buenos y malos ratos gracias por estar ahí. En particular a Andión, me has aguantado y escuchado como ninguno. A Juanito, esos ratos divertidos de aislamiento mutuo e inspiración no se olvidan. A Xeve, no tiene precio el ofrecerte y dedicar tu tiempo echando una mano en este momento tan importante que vives. Y a Ernesto, no nos hemos visto lo que me gustaría últimamente pero siempre que nos vemos parece que no pasa el tiempo.

Si continuo no acabo por lo que finalizo diciéndoos a todos los que habéis estado cerca que sois muy grandes.

¡GRACIAS A TODOS!

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ECUACIONES

Índice

RESUMEN.

En este proyecto se desarrolla un sistema electrónico para variar la geometría de un motor de un monoplaza que participa en la competición Fórmula SAE.

Fórmula SAE es una competición de diseño de monoplazas para estudiantes, organizado por "Society of Automotive Enginners" (SAE). Este concurso busca la innovación tecnológica de la automoción, así como que estudiantes participen en un trabajo real, en el cual el objetivo es obtener resultados competitivos cumpliendo con una serie de requisitos.

La variación de la geometría de un motor en un vehiculo permite mejorar el rendimiento del monoplaza consiguiendo elevar el par de potencia del motor. Cualquier mejora en del vehículo en un ámbito de competición puede resultar determinante en el desenlace de la misma.

El objetivo del proyecto es realizar esta variación mediante el control de la longitud de los tubos de admisión de aire o "runners" del motor de combustión, empleando un motor lineal paso a paso. A partir de la información obtenida por sensores de revoluciones del motor de combustión y la posición del acelerador se debe controlar la distancia de dichos tubos. Integrando este sistema en el bus CAN del vehiculo para que comparta la información medida al resto de módulos.

Por todo esto se realiza un estudio aclarando los aspectos generales del objetivo del trabajo, para la comprensión del proyecto a realizar, las posibilidades de realización y adquisición de conocimientos para un mejor desarrollo.

Se presenta una solución basada en el control del motor lineal paso a paso mediante el microcontrolador PIC32MX795F512-L. Dispositivo del fabricante Microchip con una arquitectura de 32 bits. Este dispone de un módulo CAN integrado y distintos periféricos que se emplean en la medición de los sensores y actuación sobre el motor paso a paso empleando el driver de Texas Instruments DRV8805.

Entonces el trabajo se realiza en dos líneas, una parte software de programación del control del sistema, empleando el software de Microchip MPLABX IDE y otra parte hardware de diseño de una PCB y circuitos acondicionadores para la conexión del microcontrolador, con los

sensores, driver, motor paso a paso y bus CAN. El software empleado para la realización de la PCB es Orcad9.2/Layout. Para la evaluación de las medidas obtenidas por los sensores y la comprobación del bus CAN se emplea el kit de desarrollo de Microchip, MCP2515 CAN Bus Monitor Demo Board, que permite ver la información en el bus CAN e introducir tramas al mismo.

ABSTRACT.

This project develops an electronic system to vary the geometry of a car engine which runs the Formula SAE competition.

Formula SAE is a design car competition for students, organized by "Society of Automotive Engineers" (SAE). This competition seeks technological innovation in the automotive industry and brings in students to participate in a real job, in which the objective is to obtain competitive results in compliance with certain requirements.

Varying engine's geometry in a vehicle improves car's performance raising engine output torque. Any improvement in the vehicle in a competition field can be decisive in the outcome of it.

The goal of the project is the variation by controlling the length of the air intake pipe or "runners" in a combustion engine, using a linear motor step. For these, uses the information gathered by speed sensors from the combustion engine and by the throttle position to control the distance of these tubes. This system is integrated in the vehicle CAN bus to share the information with the other modules.

For all this is made a study to clarify the general aspects of the project in order to understand the activities developed inside the project, the different options available and also, to acquire knowledge for a better development of the project.

The solution is based on linear stepper motor control by the microcontroller PIC32MX795F512-L. Device from manufacturer Microchip with a 32-bit architecture. This module has an integrated CAN various peripherals that are used in measuring the performance of the sensors and drives the stepper motor using Texas Instruments DRV8805 driver.

Then the work is done in two lines, first, control programming software system using software MPLABX Microchip IDE and, second, hardware design of a PCB and conditioning circuits for connecting the microcontroller, with sensors, driver stepper motor and CAN bus. The software used to carry out the PCB is Orcad9.2/Layout. For the evaluation of the measurements obtained by the sensors and CAN bus checking is used Microchip development kit, MCP2515

CAN Bus Monitor Demo Board, that allows you to see the information on the CAN bus and enter new frames in the bus.

1. INTRODUCCIÓN.

1.1. Fórmula SAE.

Fórmula SAE es una competición de diseño de monoplazas para estudiantes, organizado por "Society of Automotive Enginners" (SAE). El concurso se inició en 1978 y originalmente se llamo SAE Mini Indy. Actualmente existen más competiciones que siguen la línea de está que busca la innovación tecnológica de la automoción, así como que estudiantes participen en un trabajo real, en el cual el objetivo es obtener resultados competitivos cumpliendo con una serie de requisitos.

1.2. Resumen de la normativa.

El equipo debe estar formado íntegramente por estudiantes universitarios activos (incluidos los conductores). Y serán los estudiantes los que se encarguen de realizar todo el diseño del vehículo, también se encargan de la recaudación de fondos y su construcción. Siendo permitido recibir consejo de la persona que el alumno quiera.

El motor debe ser de ciclo Otto (cuatro tiempos), con una cilindrada máxima de 610cc. Se debe poner un filtro de aire limitador de potencia, para mantener la potencia por debajo de los 100 cv.

La suspensión no tiene restricciones salvo por la seguridad.

Existen pocas limitaciones en cuanto a la aerodinámica. La mayoría de los equipos no suelen trabajar mucho en este aspecto, debido a la falta de test a los que se podrían someter normalmente este tipo de componentes y este hecho está muy mal valorado por los jueces, además la velocidad máxima rara vez supera los 100 km/h.

No hay ninguna restricción de peso. El peso medio de los coches competitivos es por lo general menor de los 150 kg. Sin embargo la falta de regulación de peso junto con el techo de potencia anima a los competidores a adoptar innovadoras estrategias de reducción de peso. En 2009 se aumentó la cantidad de puntos asignados a la economía de combustible en la prueba de resistencia, esto marca una tendencia a la reducción de peso y así ahorrar en combustible.

La mayoría de las regulaciones se refieren a la seguridad. Los coches deben tener dos arcos antivuelco de acero de un determinado espesor y una determinada aleación, independientemente

de la composición del resto del chasis. Debe haber un atenuador de impactos en la parte frontal del coche, y las pruebas de este atenuador deben ser presentadas antes de la competición. Los coches también deben tener dos circuitos de freno hidráulico. Debe asegurarse que durante las pruebas el coche no pierda ningún tipo de liquido, y no debe haber una línea de visión entre el piloto y el combustible, refrigerante o aceite. Y se deben cumplir unas especificaciones para la cabina del conductor.

1.3. Pruebas de la competición.

Para poder participar en la competición se ha de superar primero una inspección técnica que no otorga puntos. Una vez superada se puede participar en las pruebas, que están divididas en pruebas estáticas y pruebas dinámicas. Cada prueba tiene una determinada puntuación, pudiéndose acumular hasta 1000 puntos como máximo.

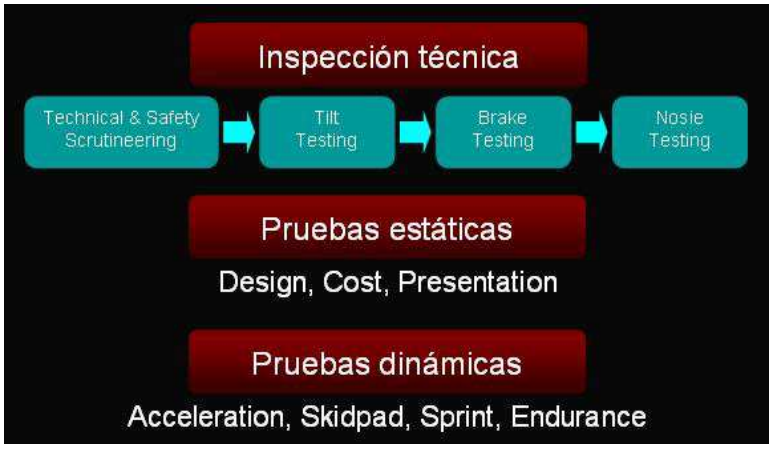


Figura 1.1: Pruebas de la competición.

Las pruebas estáticas se dividen en tres, diseño (150 puntos), coste (100 puntos) y presentación (75 puntos). Cada una de ellas tiene una determinada puntuación. Las pruebas se pueden realizar varias veces, dentro del horario establecido para cada prueba. Cuando se completen todas las pruebas estáticas el vehículo esta listo para participar en las pruebas dinámicas.

Las pruebas dinámicas son 4:

- ***Aceleración:*** El objetivo es evaluar la capacidad de aceleración del vehículo en una línea recta sobre una superficie plana. El coche debe acelerar desde parado en una distancia determinada. En función del tiempo realizado y los tiempos de los rivales se calcula la puntuación en la prueba, hasta un máximo de 75 puntos.

- ***Skidpad:*** Esta prueba busca medir la capacidad que tiene el coche para realizar giros de radio constante en forma de ocho y de esta forma evaluar la maniobrabilidad del vehículo. Cada coche puede participar en dos tandas, cada vez con un piloto distinto y con dos posibilidades en cada una de estas tandas. La puntuación de esta prueba es como máximo de 75 puntos.

- ***Sprint:*** En esta prueba el objetivo es evaluar el comportamiento del monoplaza en un circuito cronometrado con muchas curvas y sin ningún competidor más corriendo al mismo tiempo en el circuito. En esta prueba se miden distintas capacidades del coche, aceleración, freno y paso por curvas. Al igual que en las pruebas anteriores se pueden realizar dos tandas con pilotos distintos. La prueba tiene una puntuación máxima de 100 puntos.

- ***Endurance and Fuel Economy:*** En esta prueba se evalúa la concepción global de todo el vehículo sobre todo la fiabilidad y el consumo de combustible, que se medirá a la vez que se realiza la prueba de resistencia. Durante la realización de esta carrera unos 22 km no se puede repostar ni reparar el vehículo. A mitad de carrera se realiza el cambio de piloto. La puntuación de esta prueba está repartida entre el tiempo realizado y el consumo siendo 325 y 100 puntos respectivamente.

1.4. Equipo UPM-Racing.

El equipo UPM RACING se creó en la UPM para que la universidad politécnica participase en la competición anual de monoplazas Fórmula SAE.

A finales del 2003, por iniciativa del Instituto Universitario de Investigación del Automóvil (INSIA) y de la Escuela Técnica Superior de Ingenieros Industriales de la Universidad Politécnica de Madrid (UPM) y contando con el apoyo del Máster en Ingeniería en Automoción del INSIA, se constituyó el equipo UPM RACING, primer representante español en el evento. El cual, ha participado en todas las competiciones organizadas por la SAE en UK desde entonces. Hasta la fecha de hoy se han construido tres versiones del coche.

El equipo se compone de más de 50 alumnos de la Escuela Técnica Superior de Ingenieros Industriales de la UPM, del Máster de Ingeniería en Automoción del INSIA, de la escuela de Ingeniería Técnica Aeronáutica y de EUIT Telecomunicaciones. Se encuentra estructurado en 7 divisiones:

- *Aerodinámica*
- *Chasis*
- *Dinámica Vehicular*
- *Frenos*
- *Electrónica*
- *Motor-Marketing*

Cada división está liderada por un responsable que se encarga de coordinar y gestionar el trabajo de todos sus miembros, tomar decisiones técnicas y actuar como portavoz en las reuniones técnicas generales. Y por encima de estas divisiones se encuentra una dirección técnica que aporta su experiencia para tomar decisiones que afectan a varias divisiones o que los miembros de una determinada división no sepan resolver.

Participar en la FÓRMULA SAE supone un gran reto para los estudiantes que deben organizarse como un equipo de profesionales donde, además de diseñar y fabricar un monoplaza, deben administrar los recursos de que dispongan. Estos recursos deben obtenerlos por sí mismos. Igualmente, los propios alumnos pilotarán el coche y se encargarán de sus reparaciones. En resumen, se trata de formar un grupo de estudiantes, organizados como una empresa, que tiene un objetivo: medirse con las mejores universidades del mundo.

1.5. Características técnicas del coche.

El equipo, a lo largo de su existencia, ha construido varios modelos de vehículo para la competición, desde el UPM – 001 hasta el UPM – 006 que es con el que en la actualidad participa en FÓRMULA SAE. Las características técnicas del coche UPM – 006 son:

DIMENSIONES	FRENTE	TRASERA
Longitud total (mm)	2885	
Peso total incluido conductor de 68 Kg. (Kg.)	124	152

Tabla 1.1: Dimensiones del UPM – 006.

SUSPENSION	FRENTE	TRASERA
Tamaño del neumático y tipo de compuesto	20.5x6-13 R25B Hoosier	20.5x6-13 R25B Hoosier
Ruedas	Keizer aleación de 13" x6"	Keizer aleación de 13" x7''
Extensión del amortiguador. Chasis al centro de la rueda (Ns/mm)	1.35	1.78

Tabla 1.2: Suspensión del UPM – 006.

SISTEMA DE FRENOS	FRENTE	TRASERA
Rotores (incl. diámetro exterior e interior [mm] de fricción de la superficie)	Diseño de un estudiante, corte láser, montado en eje, diámetro 220 mm.	Diseño de un estudiante, corte láser, montado en eje, diámetro 220 mm.
Pinzas del freno (incl. tamaño mm)	2 x AP Racing CP 4226 ^ 2 pistones de aluminio de (25.4 mm)	2 x AP Racing CP 4226 ^ 2 pistones de aluminio de (25.4 mm)
Rodamientos del eje	61814-2RS	61814-2RS

Tabla 1.3: Sistema de frenos del UPM – 006.

MOTOR	
Fabricación/modelo	2005 Yamaha R6
Inducción	Atmosférica
Combustible	Gasolina 98 octanos
Potencia máxima	11000 rpm
Par máximo	7000 rpm
Min. RPM para el 80% del par máximo	4000 rpm
Sistema de combustible	Sistema de inyección de combustible diseñado / construido Estudiante con PE-ECU
Sensores del sistema de combustible	Disparador magnético manivela, temperatura del agua, Pos. Acelerador, MAP, Temperatura del aire. Presión y temperatura, de aceite y combustible
Volumen de admisión	3600cc
Longitud de los tubos de admisión	340mm
Escape	Cabeza de acero inoxidable
Sistema de encendido	Controlado por la ECU, lost spark system
Sistema de engrase	Carter húmedo
Deposito gasolina, ubicación, Tipo	Deposito de construcción propia localizado detrás del asiento
Silenciador	Silenciador comercial adaptado

Tabla 1.4: Motor del UPM – 006.

CHASIS	
Construcción del chasis	Marco de acero inoxidable y fibra de carbono
Rigidez torsional	1200 Nm/grado

Tabla 1.5: Chasis del UPM – 006.

Figura 1.2: Equipo UPM Racing y coche UPM – 006.

1.6. Variar la geometría de un motor.

La variación de la geometría del motor, que es el objetivo global de este proyecto, no es otra cosa que variar la respuesta de potencia y par que dicho motor es capaz de entregar en función de las rpm a las que esta trabajando. Con el fin de comprender el alcance de este objetivo es necesario comprender bien los conceptos de los términos **"par"** y **"potencia"** que, en definitiva, son dos indicadores distintos del funcionamiento del motor, pero que trabajan de la misma forma tanto en motores de combustión como en motores eléctricos; por lo que en le apartado 2.1.2 se profundizará mas en estos dos términos para mejorar su comprensión.

La finalidad global del proyecto es conseguir que las respectivas curvas de "par" y "potencia" propias y, en un principio inmóviles de un motor se conviertan en unas curvas móviles, de tal forma que el motor sea capaz de mejorar su rendimiento a lo largo de todo el rango de rpm en las que el motor es capaz de funcionar. Sin embargo, está comprobado que cuando se realiza un determinado ajuste en alguna parte del motor para mejorar su rendimiento a unas determinadas revoluciones, este ajuste, produce en otro rango de revoluciones un claro empeoramiento del comportamiento del motor, ya que los motores de combustión, en función de sus revoluciones, pueden necesitar más o menos combustible o aire o una velocidad de vaciado y llenado de los cilindros y lo usual es que un ajuste que mejora el rendimiento en bajas revoluciones no es tan bueno o, mejor dicho, empeora el comportamiento del motor a altas revoluciones.

Entonces se puede decir que la geometría de un motor viene definida por mezcla de aire y gasolina, la forma en que se quema en el interior de los cilindros y la forma en que se renuevan los gases en el interior de los cilindros. Y en función de cómo se manejen estos tres factores se obtiene una geometría de motor diferente, ya que lo normal es que para un determinado motor se adapte la mezcla de aire-combustible, forma de quemado y el vaciado y llenado de los cilindros para que el motor en la curva de par (y potencia) entregue la mejor respuesta buscada por el objetivo final del motor. Así por ejemplo si el motor es para un tractor, el motor tendrá un alto par en bajas revoluciones; sin embargo, si se trata de un motor para un vehículo familiar, lo que se buscara es que el motor tenga una respuesta plana de potencia a lo largo de todo el rango de revoluciones; y, para el caso de, por ejemplo, un vehículo rápido el motor tendrá una buena respuesta de potencia a altas revoluciones.

Otra característica asociada la geometría del motor es el aprovechamiento del combustible; cuando el motor está por debajo del punto de potencia máxima se puede decir que el motor no está trabajando a pleno rendimiento y por lo tanto se desperdicia combustible, por el contrario si

las revoluciones se llevan por encima de este punto de potencia máxima, el motor puede que esté intentando girar más rápido de lo que debiera impidiendo que el combustible se queme apropiadamente. Es decir, cuando el motor se mantiene en el punto máximo de potencia es cuando la relación consumo / prestaciones del motor es la mejor, cuando se va por encima o por debajo de este punto se está desperdiciando combustible.

Una vez se ha comprendido que es la geometría de un motor se entiende porque resulta tan interesante tener un sistema que varíe la geometría del motor en un monoplaza de fórmula SAE ya que se mejora su respuesta en potencia a la vez que se mejora el consumo de combustible que son dos características evaluables en las pruebas de la competición.

1.7. Formas de variar la geometría de un motor.

Como se ha venido comentado la geometría de un motor responde principalmente a tres aspectos, dependiendo de cual de estos aspectos se modifiquen sobre el motor se variara su geometría si bien existen diversas formas de realizar esta variación sobre cada uno de estos aspectos del motor aquí se comentaran tan solo tres a modo de ejemplo:

1.7.1. Sistema de distribución variable.

Este sistema consigue variar los tiempos de llenado y vaciado de los cilindros del motor, es decir, hace variar el tiempo de apertura y cierre de las válvulas de admisión y escape de los cilindros del motor, en función del régimen de vueltas y la carga del motor se optimiza el proceso de renovación del combustible en los cilindros del motor.

La proporción de mezcla aire/combustible que entra comparada con la que podría entrar depende del tiempo disponible en el ciclo de abrir y cerrar las válvulas de admisión y escape. Con objeto de dinamizar este proceso, hay un momento en que las válvulas de admisión y escape están abiertas a la vez, se conoce como “cruce de válvulas”. Estas válvulas controlan el flujo de admisión y escape de los gases en la cámara de combustión. El tiempo, la duración y la elevación del ciclo de abrir y cerrar de la válvula tienen un impacto significativo en la geometría del motor.

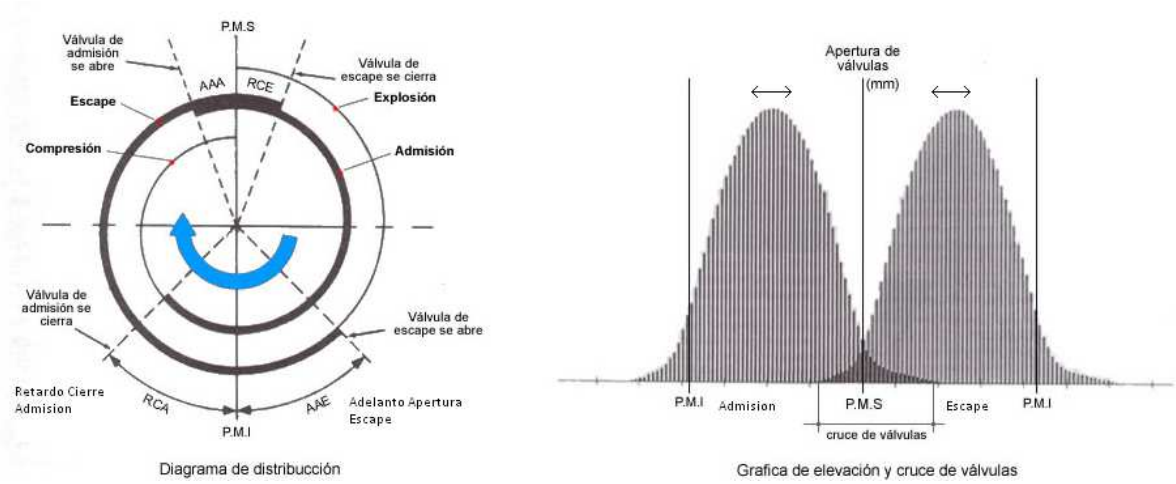


Figura 1.3: Variación de los tiempos de apertura y cierre de válvulas.

Un motor equipado con un sistema de distribución variable permite mejorar el rendimiento en el rango de funcionamiento del motor. En la zona de bajas rpm, un cruce reducido favorece un ralentí estable y unas emisiones bajas. En altas rpm, el poco tiempo entre ciclos Otto requiere un mayor periodo de cruce de válvulas. Al modificarse los tiempos de apertura y cierre de las válvulas varía el llenado de mezcla aire combustible, obteniendo lo mejor de las dos situaciones en el comportamiento del motor para que haya un mayor aprovechamiento del combustible, menores emisiones y máximo par motor.

Este sistema varia, normalmente, por presión hidráulica la posición del árbol de levas de las válvulas de admisión en relación con el cigüeñal del motor. La posición del árbol de levas puede avanzarse o retrasarse una cierta cantidad de grados respecto al cigüeñal, esta cantidad de grados depende del motor, longitud de válvulas, pistón, etc. En la figura a la derecha el árbol de levas ha variado unos grados su posición lo que hace variar los tiempos de apertura de las válvulas.

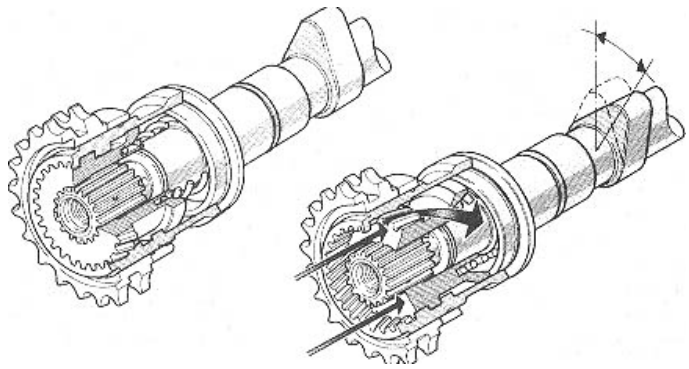


Figura 1.4: Desviación del árbol de levas.

En la actualidad, el sistema VVT-i (Variable Valve Timing) ofrece una distribución variable continua y permite que la unidad de control de motor especifique la distribución óptima de acuerdo con las condiciones de manejo.

1.7.2. Escape de gases variable.

Este sistema es un ejemplo de variación de la velocidad de vaciado de los gases del motor, último tiempo del ciclo Otto que iría acompañando de un ajuste previo en la distribución del motor. El EXUP (EXhaust Ultimate Power valve) un dispositivo incorporado en algunas motocicletas de Yamaha que ajusta constantemente el diámetro interno del sistema de escape en consonancia a las revoluciones del motor, consiguiendo, así, un buen rendimiento y una entrega lineal de potencia en toda la gama de revoluciones.

El sistema consiste en la instalación de una válvula en el interior del escape en el punto donde se juntan los colectores del escape. Para obtener el mejor rendimiento del motor a altas revoluciones, el sistema de distribución se diseña para que las válvulas de escape de los cilindros se anticipen en su apertura, lo que es beneficioso en ese régimen del motor, pero es contraproducente a bajas revoluciones. Cerrando en este caso la válvula del sistema EXUP, se crea una presión de retorno dentro del sistema de escape, que compensa el efecto del adelantamiento de las válvulas de escape de los cilindros. Como consecuencia se logra una entrega más lineal de potencia y una gran entrega de potencia a altas revoluciones.

Un servomotor controlado por el modulo de encendido abre o cierra el EXUP, que pasa de estar completamente cerrado al ralentí a una apertura total a 9000 rpm.

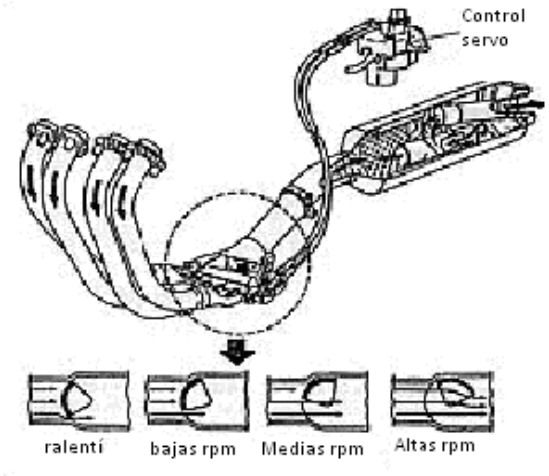


Figura 1.5: Sistema EXUP.

1.7.3. Admisión variable.

Durante el primer tiempo del ciclo Otto el pistón inicia su carrera de descenso, se aprovecha esta succión que se crea para introducir la mezcla aire/combustible en el cilindro, para que se llene mejor el cilindro aprovechando la inercia de los gases existe un periodo en el que las válvulas de admisión y escape de un motor permanecen abiertas a la vez, ver figura 1.3.

Creando distintas turbulencias se busca mejorar este llenado del cilindro sin modificar la distribución del motor. Algunos motores tienen mecanismos en las piezas que conducen el aire de admisión para variar el recorrido que hace el aire antes de entrar en los cilindros, otros varían la longitud del tubo por donde circula el aire de admisión hacia el cilindro, otros el volumen del colector del que toma el aire cada cilindro. El propósito es el mismo, adecuar la frecuencia de los pulsos del aire de admisión a distintos regímenes del motor. La mayoría de los colectores variables, que es como se los conoce, tienen dos modos de funcionamiento, aunque los hay con más modalidades.

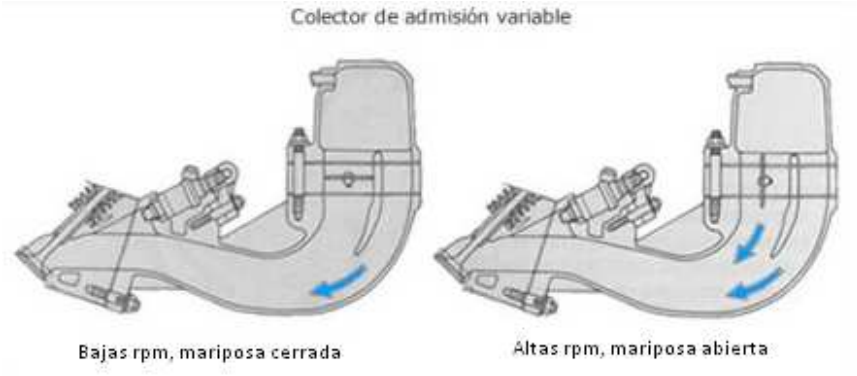


Figura 1.6: Sistema de admisión variable.

1.8. Proyecto UPM – Racing.

El proyecto que se presenta consiste en variar la geometría del motor del monoplaza mediante un sistema de admisión variable. Esto se consigue mediante el control de la longitud que tienen los “runners”, es decir, los conductos de entrada de aire al cilindro.

La longitud de estos “runners” permite crear un remolino de aire en la cámara de combustión, que ayuda a distribuir de forma homogénea la mezcla aire/combustible. A bajas revoluciones, se incrementa la velocidad del flujo de aire, al dirigirlo a través de un camino más largo con una capacidad de entrada limitada, esto mejora el par a baja velocidad del motor. En altas

revoluciones interesa que el camino sea lo más corto posible para que mayor cantidad de aire con menor resistencia puedan entrar en la cámara de combustión. Además, el sintonizado de la distancia de la admisión con el remolino de entrada de aire, en resonancia provoca un efecto de presurización, lo que aumenta el rendimiento del motor. Similar a un compresor de baja presión.

Figura 1.7: Esquema de la admisión.

Entonces se necesita el diseño de un control electrónico capaz de actuar sobre el soporte de los "runners" cambiando la longitud de estos en función de las revoluciones por minuto del motor y de la posición del acelerador. Esto es el objeto del proyecto.

Desde el INSIA se han facilitado los siguientes datos y especificaciones para la realización del proyecto:

- El rango de revoluciones a controlar va desde las 8000 rpm hasta las 15000 rpm.
- La longitud de los tubos de admisión ("runners") varía entre 150mm a 330mm para el rango de revoluciones a controlar.
- Se ha indicado que se desea tener el control de la distancia a lo largo de rangos de revoluciones sabiendo de esta forma que entre las 8000 y las 9000 rpm la longitud del "runner" es de 160mm, de tal forma que la distancia total a recorrer se hace en escalones y la longitud del "runner" no cambia hasta superar un umbral, evitando el ajuste infinito.
- Además llegado el momento es interesante poder modificar estos tramos para que se pueda probar el comportamiento del monoplaza.
- La distancia de los "runners" se incrementa cuando se ha actuado un 65% sobre el acelerador.
- Conexión al bus CAN, parta transmitir rpm del monoplaza.

El proyecto, por su parte, adelanta que existe un sensor de posición del acelerador, el TPS (Throttle Position Sensor), es un sensor ligado al pedal del acelerador con una señal de 0 a 5V,

que indica el porcentaje de acelerador que esta actuando. Siendo 0 V sin carga en el acelerador y 5 V para el acelerador con una carga del 100%.

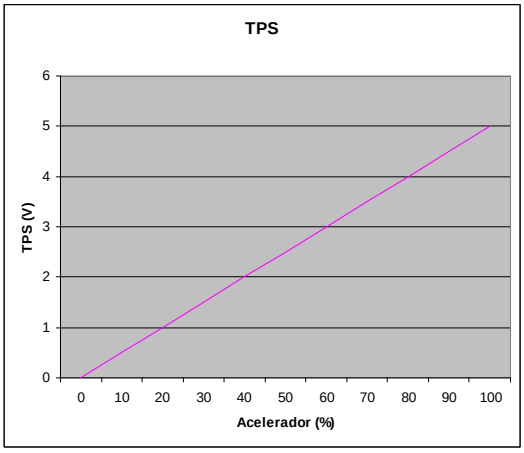
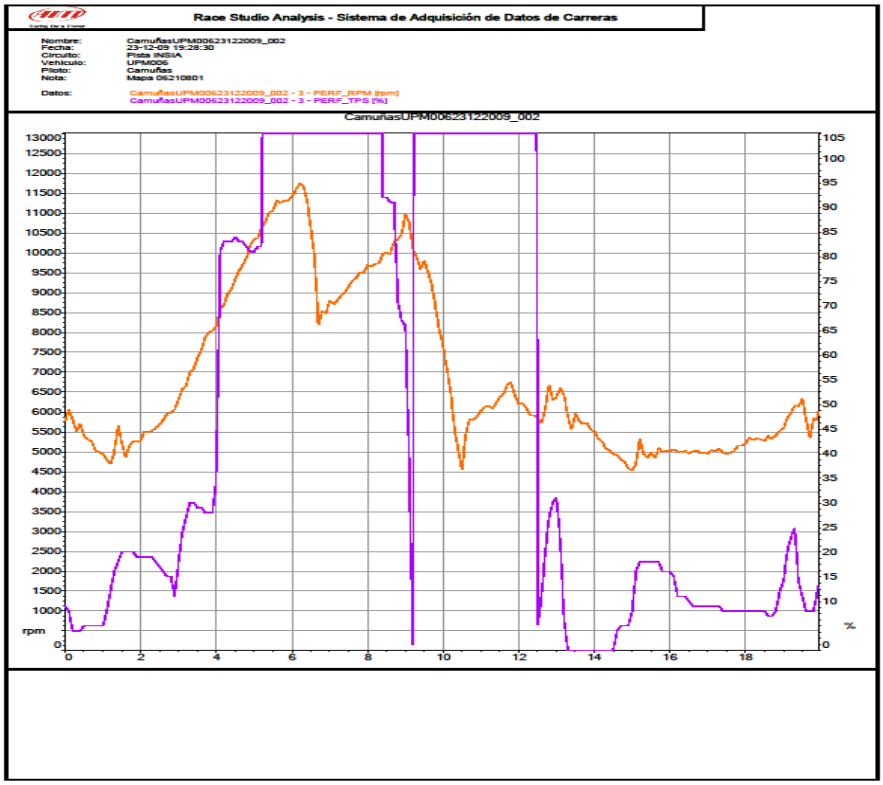


Figura 1.8: Señal TPS.

También se ha facilitado información sobre la evolución de las revoluciones del motor a lo largo del tiempo cuando se esta actuando sobre el acelerador al 100%



Morado: posición TPS (der.)

Naranja: rpm motor (izq.)

Eje x: tiempo (s)

Figura 1.9: Gráfica de evolución del motor.

Como se puede ver en la figura 1.9 las revoluciones por minuto del motor crecen muy rápido, por lo que será necesario variar esta distancia igual de rápido. Desde el INSIA se facilitó la comprensión de esta gráfica indicando que para un correcto funcionamiento del sistema en el vehiculo la distancia del "runner" debe recorrerse en su totalidad en un tiempo de un segundo, de este modo se desea que la pieza que se encargue de tirar o empujar los "runners" debe desplazarse a una velocidad minima de 0,2 m/s. Así mismo se ha indicado que este "bloque" tirador mas los 4 "runners" que dispone el motor del UPM – 006 tienen un peso de 400 g.

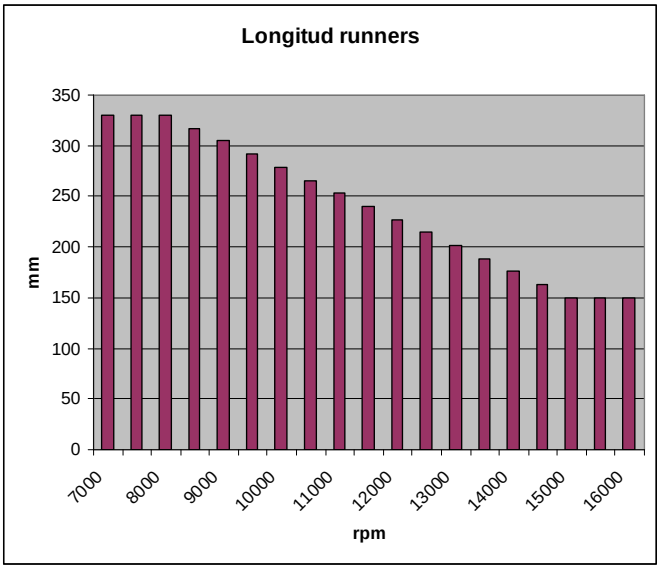


Figura 1.10: Ejemplo de longitud de "runners".

Además el dispositivo de control debe conectarse a un bus CAN existente en el monoplaza. En cuanto al formato de datos no se ha indicado nada.

2. Motores de DC.

2.1. Introducción a motores DC.

2.1.1. Principio de funcionamiento.

La producción de un "par-motor" en un motor eléctrico, requiere disponer de un campo magnético en el que colocar dentro unos conductores eléctricos, al hacer pasar corriente por el conductor, éste reacciona con el campo magnético y genera una fuerza que será la responsable de producir el par y, por tanto, el movimiento del motor. En su ecuación "B" corresponde al flujo del campo magnético, "i" a la corriente que circula por el conductor y "L" a la longitud del mismo, siendo su expresión:

$$F = B * i * L$$

Ecuaciones 2.1: Fuerza generada por un inductor.

El campo magnético necesario se produce por medio de los polos magnéticos (Norte, Sur), los cuales son creados por medio de una bobina arrollada sobre un núcleo ferromagnético. Normalmente se hace referencia a estas bobinas como Campo.

Los conductores son la otra parte necesaria para la producción del par están colocados también en un núcleo ferromagnético móvil, para facilitar su giro al producirse la fuerza sobre ellos. A esta parte se la llama armadura.

Los generadores de DC, actúan de forma similar, se tiene un campo magnético y dentro de él los conductores eléctricos, los cuales se hacen girar cortando el flujo, con lo cual se inducirá sobre ellos una fuerza electromotriz.

El modelo general de un motor puede representarse por la relación de la entrada, donde ésta puede ser la tensión, con la salida, que podrá ser el desplazamiento angular del eje del motor. De esta manera se estarán modelando, a la vez, la parte electromecánica, la parte mecánica y la relación entre ambas.

Estos motores se pueden controlar de dos formas, controlando su campo y manteniendo constante la corriente por la armadura o controlando la armadura y manteniendo constante su corriente de campo.

Existen dos grandes tipos de motores de DC, los motores DC propiamente, que giran de forma continuada y motores paso a paso, que giran según una secuencia de excitación aplicada en sus bobinas.

2.1.2. El par y la potencia de un motor.

El par (también llamado troque) y la potencia son dos indicadores del funcionamiento del motor, y señalan qué tanta fuerza puede producir y con qué rapidez puede trabajar.

Se llama troque o par máximo a la mayor cantidad de fuerza de giro que puede desarrollar el motor. Esto sucede a cierto número de revoluciones. Si se sigue el ejemplo de la gráfica inferior, se ve que el motor tiene un torque máximo de 120 Nm a 2500 rpm, esto significa que el motor produce una fuerza de giro máxima (conocido como momento o par torsional) de 120 Nm cuando esta acelerado al máximo y gira a 2500 rpm.

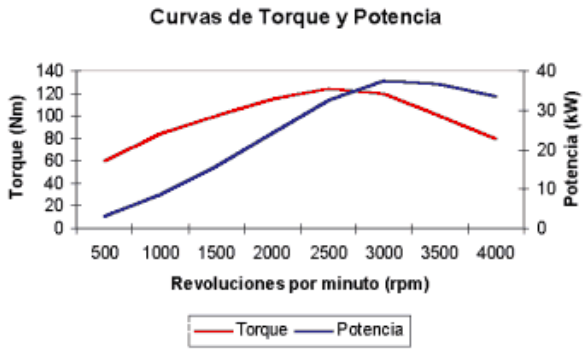


Figura 2.1: Gráfica par y potencia.

El par motor es la fuerza que producen los cuerpos en rotación, es decir, el momento de fuerza que ejerce un motor sobre el eje de transmisión de potencia, la potencia desarrollada por el par motor es proporcional a la velocidad angular del eje de transmisión, que viene dada por la siguiente ecuación, donde P es la potencia (W), M es el par motor (Nm) y ω es la velocidad angular (rad/s).

$$P = M * \omega$$

Ecuaciones 2.2: Cálculo de par.

La potencia indica la rapidez con que puede trabajar el motor, siendo la potencia máxima el mayor valor que se obtiene de multiplicar el par del motor por la velocidad de giro.

Un ejemplo práctico para comprender la diferencia entre par y potencia es posible tomarle del ejercicio físico de montar en bicicleta. El motor sería la persona que pedalea y el par motor el esfuerzo ejercido por el par de fuerzas que se aplica sobre los pedales. Con un determinado pedaleo de la bicicleta se consigue una determinada velocidad fija; digamos que, dando 30 giros o pedaladas por minuto y aplicando una determinada potencia, se llega a una velocidad de 15km/h. Ahora bien, si cambia de piñón, y reduce a 15 los giros o pedaleadas por minuto, pero generando la misma potencia, se tendría que poner el doble de par motor, puesto que tendría que ejercer el doble de fuerza en la pedalada, para poder mantener la velocidad de 15 km/h.

2.2. Motor paso a paso.

2.2.1. Descripción.

El motor paso a paso es un dispositivo electromecánico que convierte una serie de impulsos eléctricos en desplazamientos angulares discretos, lo que significa que es capaz de avanzar una serie de grados (pasos) dependiendo de sus entradas de control. El motor paso a paso se comporta del mismo modo que un conversor digital / analógico y puede ser controlado por impulsos procedentes de sistemas lógicos.

El motor paso a paso se compone de un imán permanente en el rotor y dos pares de bobinados en el estator. El imán de polos permanentes se alinea y monta con el eje del rotor y cada uno de los polos tiene que disponer de dientes en forma escalonada, con el fin de poder producir a su vez muchos polos. Ahora bien, los polos del estator del motor paso a paso también tienen dientes, los dientes del rotor y el estator se colocan de tal modo que exista un desfase entre los polos de las dos fases; este desfase es conocido como ángulo de paso.

Ventajas:

- Fácil de colocar, pues se mueve por pasos sobre la base de pulsos aplicados a los bobinados del estator.
- El sentido de giro se cambia al invertir la base de pulsos.
- La velocidad es controlada por la frecuencia de la base de pulsos o impulsos.
- Los motores paso a paso tienen un par de mantenimiento alto, pero no pueden funcionar a alta velocidad.

Inconvenientes:

- La máxima frecuencia admisible es tan elevada como la de un motor DC. Pero, si la frecuencia de pulsos es demasiado elevada, el motor puede reaccionar erróneamente o puede incluso, no realizar ningún movimiento; es decir, vibrar pero sin llegar a girar o girar erráticamente.

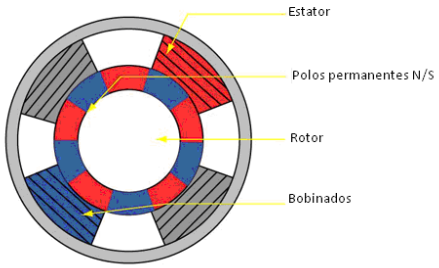


Figura 2.2: Partes de un motor paso a paso.

2.2.2. Clases.

Se pueden destacar dos grandes clases dentro de los motores paso a paso los motores unipolares que se caracterizan por ser los más simples de controlar ya que la corriente por las bobinas siempre circula en el mismo sentido y los motores bipolares que necesitan ciertos trucos para ser controlados debido a que requieren del cambio de dirección de flujo de corriente a través de las bobinas en la secuencia apropiada para realizar el movimiento, por lo que un mismo extremo del bobinado puede tener en uno de sus extremos distinta polaridad (bipolar).

Según su tecnología el motor paso a paso puede ser: El motor de paso de rotor de imán permanente que permite mantener un par diferente de cero cuando el motor no está alimentado, el motor de paso de reluctancia variable tiene un rotor multipolar y un estator devanado laminado, y rota cuando los dientes del rotor son atraídos a los dientes del estator electromagnéticamente activos y el motor híbrido que es una mezcla de los tipos de reluctancia variable e imán permanente. Este tipo de motor tiene una alta precisión y alto par.

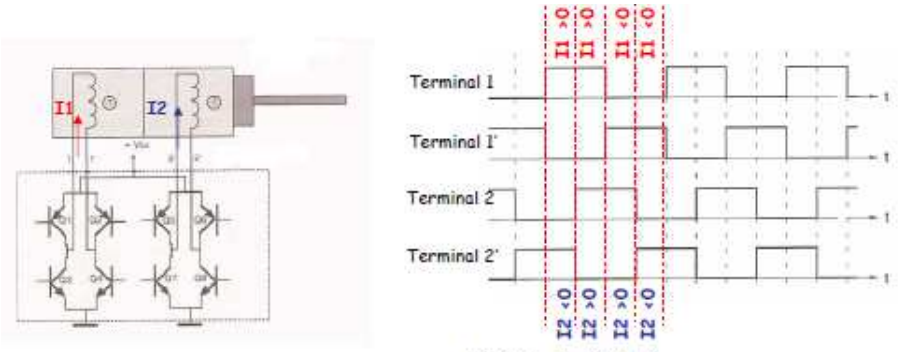
2.2.3. Funcionamiento y control.

En términos simples, el rotor del motor paso a paso se compone de imanes permanentes y los polos del estator con bobinados. El rotor está montado y construido con un solo imán en línea

con el eje del rotor, pero, justamente cuando el rotor se alinea con uno de los polos del estator, la segunda fase se activa. Las dos fases alternan encendido y apagado, y además invierten la polaridad, con lo que la variación de la dirección del campo magnético creado en el estator producirá movimiento de seguimiento por parte del rotor de imán permanente, el cual intentará alinearse con campo magnético inducido en las bobinas.

Para el control de un motor paso a paso del tipo bipolar, se establece la configuración de "Puente H", de forma que, activados los transistores Q1 y Q4, permiten la alimentación en un sentido que hace que el motor gire, si se cambia el sentido de la alimentación activando Q2 y Q3, se cambia el sentido de la corriente y el motor gira en sentido contrario.

La posición del motor se puede controlar con precisión sin ningún tipo de mecanismo de realimentación, un control de lazo abierto, y se mueve en pasos sobre una base de pulsos suministrados sobre los bobinados del estator. El sentido de giro se cambia al invertir la secuencia de pulsos y la velocidad está controlada por la frecuencia de la base pulsos. Un avance en el control paso a paso es incorporar una realimentación de la posición del rotor, por ejemplo, un encoder, con lo que se mejora la posición real del rotor. Esto convierte al motor paso a paso en un servomotor sin escobillas de alta resolución de la posición. Con esta técnica se va a ejecutar normalmente el control del motor en modo de lazo abierto y sólo se empleará el control en el modo de lazo cerrado si el error de posición del rotor es demasiado grande, lo que permitirá al sistema evitar oscilaciones, un problema común en los servomotores.



Paso	Q1Q4	Q2Q3	Q5Q8	Q6Q7	I1	I2
1	ON	OFF	OFF	ON	+	-
2	ON	OFF	ON	OFF	+	+
3	OFF	ON	ON	OFF	-	+
4	OFF	ON	OFF	ON	-	-

Sentido horario

Sentido antihorario

Figura 2.3: Ejemplo conexión y excitación de un motor paso a paso bipolar.

El rendimiento del motor paso a paso depende en gran medida del circuito driver. La curva de par del motor se puede recorrer con mayor velocidad si los polos del estator se pueden invertir con mayor velocidad, en cuyo caso el factor limitante es la inductancia de la bobina. Hay dos tipos significativos de circuitos driver para un motor paso a paso, el circuito de unidad L / R (tensión constante) y circuitos chopper (corriente constante).

Cada motor paso a paso tiene un ángulo de paso definido asociado a él. En el ejemplo que podemos ver con dos fases, se tiene un ángulo de paso de 90º. Existen algunas técnicas básicas que nos permiten mejorar la resolución de los motores paso a paso, disminuyendo el ángulo de paso, para aumentar la resolución del paso, se divide el paso completo en la sub-etapas (medio paso o micropaso) o doblar el ángulo del paso.

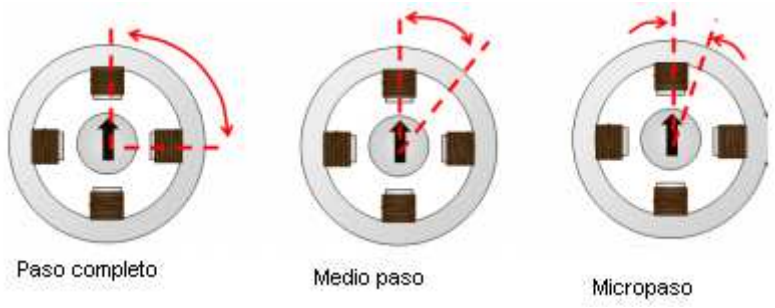


Figura 2.4: Desplazamientos de un motor paso a paso.

Dependiendo del tipo de motor paso a paso (unipolar, bipolar) se han de excitar los extremos de sus bobinas del estator según una determinada secuencia para que el motor se desplace. Haciendo cambios sobre estas secuencias se pueden conseguir distintos objetivos.

- ***Paso completo:*** Secuencia en la que un motor hace un paso normal por cada impulso de señal que recibe. En la tabla de figura 2.3 se ve el ejemplo en de una secuencia de excitación para un motor paso a paso bipolar controlado por un puente en "H". En la tabla 4.3 está la secuencia para un motor unipolar.
- ***Medio paso:*** Con esta técnica de señal se reduce el tamaño del paso, mejorando la resolución de la posición, pero empeorando su torque, En la tabla 4.4 se ve una secuencia de excitación de medio paso par un motor unipolar.
- ***Wave drive o paso simple:*** Es la secuencia de pasos más simple, consiste en ir activando las bobinas de una en una, Se consigue que el consumo de potencia del motor sea menor, pero también lo será su par. La tabla 4.5 es la secuencia de excitación para un motor unipolar.

- ***Paso doble:*** Secuencia en la que se activan las bobinas de dos en dos, aumentando el par y realizando pasos mas largo, pero mas bruscos, perdiendo resolución de posicionamiento.
- ***Micropaso:*** Este método de excitación es el mas complejo, ya que a parte de generar las secuencia adecuada, se emplea la misma secuencia que en el medio paso, se ha de controlar la corriente que circula por cada bobinado, y de este modo se consiguen multitud de pasos en función de la corriente que pase por cada bobina.

Para el control de este tipo de motores se emplean estímulos generalmente impulsos digitales. Las E/S básicas para el control del motor paso a paso son módulos Captura / Comparación / PWM. El driver empleado puede se el de varios conmutadores MOSFETs conectados al microcontrolador. Realizando transiciones suaves entre etapas se reducen los problemas de resonancia y ruido. El par máximo del motor se da cuando las tasas de pasos son muy bajas.

2.2.4. Motores lineales paso a paso.

Para la consecución del objetivo del proyecto la elección de un motor lineal paso a paso es buena opción, ya que aun al tratarse de un motor paso a paso y no ser motores muy rápidos, existen modelos capaces de desplazarse a la velocidad deseada, con la ventaja de su control, mas sencillo que el de otro tipo de motores DC. Un motor paso a paso lineal tiene un funcionamiento y unas configuraciones de conexionado iguales que los motores paso a paso convencionales, el cambio es que en los motores tradicionales el giro del rotor es el que da el movimiento, sin embargo en los motores lineales el rotor esta perforado y tiene una rosca en la cual se introduce una varilla sin fin y el motor al funcionar lo que hace es enroscar o desenroscar el rotor en la varilla, produciéndose así el desplazamiento lineal, de esta forma existe una relación entre la rosca que hay en el rotor con el ángulo de paso del motor y de esta forma se reconoce el desplazamiento lineal por paso. Según el fabricante de que se trate se detallan más o menos características. Como se ha dicho existen relaciones entre categorías y se pueden obtener algunas de estas calculándolas a partir de las más importantes, que se facilitan a continuación.

- ***Thrust:*** Es el empuje o fuerza guarda relación con el par máximo que el motor es capaz de desarrollar sin perder pasos, es decir sin dejar de responder a alguna excitación del estator. Este empuje empeora al aumentar la velocidad del motor. Se da en Newton (N).
- ***Holding force:*** Es la fuerza propia de los motores de imanes permanentes, producida por los imanes del rotor, cuando por los devanados del estator no circula corriente.

- ***Linear Travel o resolución:*** Es el desplazamiento lineal que se produce como consecuencia de un impulso de excitación en el estator. Viene dado en mm/paso.
- ***Spindle pitch:*** Es el paso del roscado que lleva el motor en el eje del rotor.
- ***Steps per revolution:*** El desplazamiento del rotor del motor lineal es igual que en los motores convencionales, el rotor gira. Con este dato se sabe el número de estímulos que hay que realizar para que el rotor de una vuelta completa.
- ***Step angle:*** Ante un estimulo del estator el rotor gira una cierta cantidad de grados, que es lo que se llama ángulo de paso. Pudiéndose calcular el número total de pasos por vuelta.

$$\alpha = \frac{360^{o}}{N}$$

α: ángulo de paso.

N: número de pasos por vuelta.

Ecuaciones 2.3: Cálculo de ángulo de paso.

- ***Maximum pull-in rate:*** Es la frecuencia máxima de impulsos que se le puede enviar al estator sin que pierda pasos. Esta magnitud viene dada en pasos por segundo. Esta información también se puede facilitar en gráficas que relacionan la frecuencia de paso con el empuje del motor.

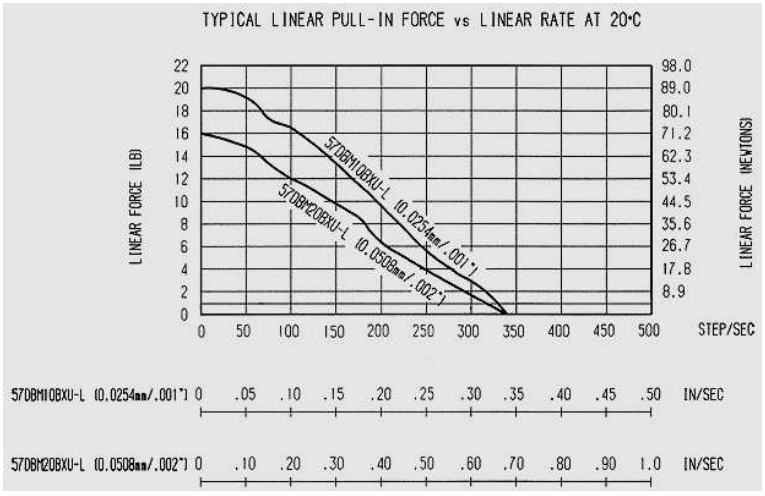


Figura 2.5: Gráfica fuerza-frecuencia de paso (Portescap56DBM10B2U-L).

2.2.5. Usos comunes.

El motor paso a paso se utiliza a menudo con un microprocesador y equipos basados en microcontroladores, ya que ofrece unos excelentes resultados en control de posición de lazo

abierto. Los motores paso a paso pueden ser utilizados en aplicaciones de instrumentación y medida, el posicionamiento de ejes múltiples, impresoras y equipo de vigilancia.

Otros posibles usos de este tipo de motores serian para implementar sistemas como:

- Ajustar la velocidad de ralentí.
- Los gases de escape de recirculación.
- Paletas conducto del flujo de aire.
- Control de espejos.
- Telescopios.
- Antenas.
- Juguetes.

2.3. Perfiles de velocidad.

En general la frecuencia de pasos de arranque de los motores paso a paso es menor que la frecuencia máxima de funcionamiento. Para alcanzar la posición final la frecuencia de pasos se reduce gradualmente hasta que el motor se detiene. El tiempo de frenado suele ser menor que el tiempo de aceleración ya que la carga ayuda a parar el motor. Al gráfico de la razón de pasos en función del tiempo en el que el motor se mueve entre la posición inicial y la posición final es lo que se conoce como perfil de velocidad.

Existen diversos métodos para generar el perfil de velocidad óptimo, o aproximaciones a este, siendo lo primordial que los parámetros del sistema y la capacidad de aceleración/deceleración del motor se estabilicen. Es importante tener en cuenta la gráfica de pull-in rate y el empuje necesario para la elección del perfil de velocidad adecuado, ya que cada motor tiene una velocidad máxima que puede alcanzar en función de la carga.

2.3.1. Partir parar.

Es el tipo más sencillo de perfil de movimiento, donde no existe ningún periodo de aceleración y de desaceleración. En este perfil de movimiento la carga se mueve de un tirón, que pasa de parada a moverse a una determinada velocidad dada por la frecuencia de impulsos. El modo partir parar es adecuado para aplicaciones en las que las características del motor estén por encima de los requerimientos del sistema, una situación que se suele buscar en el momento de

realizar un diseño empleando motores paso a paso, considerándose despreciables el tiempo de aceleración y los efectos de la carga.

La frecuencia de impulsos que se ha de aplicar al motor se obtiene de las siguientes fórmulas:

$$N\_pasos = \frac{recorrido}{mm/paso}(pasos)$$

$$Frec\_step = \frac{N\_pasos}{t}(pasos/s)$$

Ecuaciones 2.4: Perfil partir parar.

Teniendo un motor con una resolución de 25µm/paso, y un recorrido de 180 mm en 1 segundo, se obtiene una frecuencia de impulsos de 7200 (pasos/s). Este perfil de movimiento es adecuado para el proyecto si se dispone de un motor que tenga el empuje necesario para despreciar la carga, si el paso a paso no tiene el empuje necesario para mover la carga instantáneamente es posible que se pierdan pasos o que el motor no gire adecuadamente.

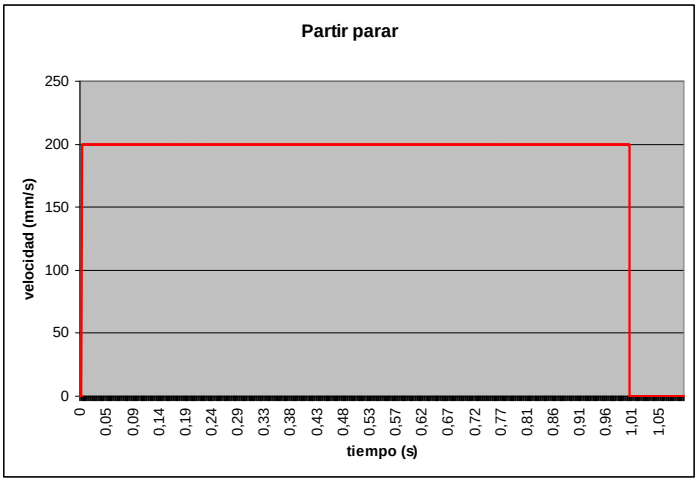


Figura 2.6: Perfil partir parar.

2.3.2. Triangular

Si la velocidad media necesaria es menos de la mitad de la velocidad máxima del motor se puede emplear el perfil triangular, teniendo una aceleración y un frenado lentos.

Si se hace que el tiempo de aceleración sea igual que el de frenado, la aceleración y la desaceleración que se tendrán se calculan de la siguiente manera:

$$t_{acc} = t_{desac} = \frac{t_{total}}{2}$$
$$v_{\max} = 2 * v_{media}(mm/s)$$
$$acc = desac = \frac{v_{\max}}{t_{acc}}(mm/s^2)$$

Ecuaciones 2.5: Perfil triangular.

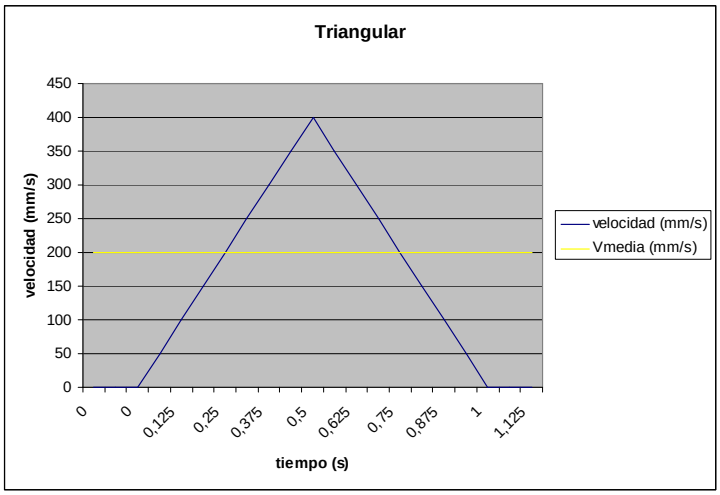


Figura 2.7: Perfil triangular.

Para realizar un movimiento a 200 mm/s en 1 segundo, se tendrá un tiempo de aceleración igual a 0.5 segundos. Dando una aceleración 800 mm/s^2. En este caso en particular, se ve que este perfil de movimiento no se puede emplear para el proyecto ya que la velocidad máxima necesaria para poder desplazarse a la velocidad media adecuada es muy alta.

2.3.3. Trapezoidal.

Cuando la velocidad de trabajo del motor paso a paso es elevada y su carga no puede considerarse ligera para el empuje que tiene el motor, se emplea el perfil trapezoidal que realiza aceleraciones y desaceleraciones controladas, permitiendo el control de la velocidad del movimiento. Este perfil también puede ser útil para la realización del proyecto ya que será necesaria su utilización en función de la masa a mover.

Usualmente se intenta acelerar lo más posible en el comienzo del movimiento, cuando el motor está parado o se mueve más despacio para aprovechar que en esos instantes su par es mayor, suavizando su aceleración hasta alcanzar la velocidad o posición objetivo, de tal forma que cuando el motor la alcance, su aceleración sea lo menor posible. También se suele emplear el mismo tiempo y rampa para aceleración que para desaceleración, ya que simplifica la obtención

del perfil de velocidad, pero todo esto esta siempre limitado por el motor. Por eso, se suelen emplear motores con prestaciones por encima de las necesidades, ya que permite un manejo mejor del motor paso a paso.

En la figura siguiente el tiempo de aceleración sería el correspondiente a la rampa de subida, le sigue el tiempo de velocidad constante, durante el cual la velocidad del motor será superior a la velocidad media compensando la menor velocidad durante los tiempos de aceleración y frenada para obtener la velocidad media deseada. Finalmente esta el tiempo de frenado durante la rampa de bajada de la velocidad.

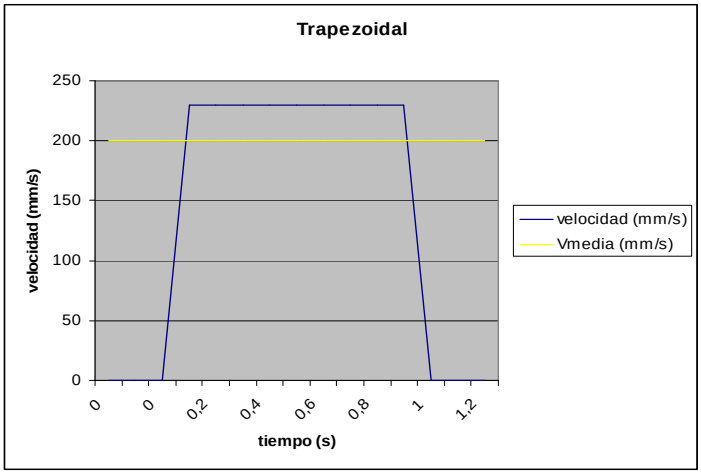


Figura 2.8: Perfil trapezoidal.

El cálculo de los tiempos de aceleración y frenado que realiza el motor se obtiene a partir de la gráfica pull-in rate (Figura 2.5) y de la inercia de la carga, obteniendo el tiempo que tarda el motor en alcanzar una frecuencia de pasos o variar de una frecuencia pasos a otra cuando esta moviendo una carga con las siguientes ecuaciones:

$$\sum F = m\frac{\partial^2 r}{\partial t^2}$$

$$\frac{\partial r}{\partial t} = LT * f$$

$$T(f) - T_L(f) = m * LT\frac{\partial f}{\partial t}$$

$$\frac{1}{m * LT}\int_0^{t1}\partial t = \int_0^{f1}\frac{1}{T(f) - T_L(f)}\partial f$$

$$t_1 = m * LT * \int_0^{f1}\frac{1}{T(f) - T_L(f)}\partial f$$

Ecuaciones 2.6: Tiempo de aceleración.

Siendo T(f) la respuesta de empuje del motor y TL(f) la inercia de la masa en función de la frecuencia de desplazamiento, m es la masa de la carga, LT es la característica Linear Travel del motor y f_1 es la frecuencia de pasos que se va a alcanzar, repitiéndose la integral, variando los límites para cada tramo de aceleración y sumándolos, para obtener el tiempo total de aceleración.

El establecimiento de la velocidad media se ha de calcular asignando tiempos de aceleración, frenado y de velocidad máxima constante para el recorrido de una distancia fijada para controlar la velocidad media dentro de todo un recorrido mayor, esta distancia es la nueva "resolución" de posición del motor, buscándose siempre que esta sea minima. Hay que decir que esta nueva "resolución" puede no ser fija dependiendo del control de motor que se haga, si se pierde esta "resolución" la velocidad media para distintas longitudes de recorrido no será constante resultando una velocidad mayor cuanto mayor sea el recorrido y una velocidad media menor en caso de que el recorrido sea menor al de la "resolución".

Para calcular la velocidad media real del desplazamiento se necesita saber cuantos pasos da en cada tramo de frecuencia; si éste no es un número entero se redondea, haciéndose de nuevo el cálculo a la inversa con el número de pasos obtenido ya redondeado para obtener el tiempo real del desplazamiento. Para obtener el tiempo total, se suman los tiempos reales de cada tramo y se calcula la distancia recorrida con el número total de pasos dados. Y al dividirse entre el tiempo total, se obtiene la velocidad media.

$$N\_pasos_1 = \frac{t_1}{f_1}$$

$$t_{1\_REAL} = N\_pasos_{R\_1} * f_1$$

$$v_{media} = \frac{\frac{mm}{paso}\sum N\_pasos_{R\_i}}{\sum t_i} \Leftrightarrow v_{diseño} = \frac{recorido\_del\_perfil}{t_{acc} + t_{dec} + t_{v\_cte}}$$

Ecuaciones 2.7: Cálculo de velocidad media.

3. Bus CAN

3.1. Bus CAN.

CAN Bus es un protocolo de comunicaciones desarrollado por la firma alemana Bosch en los años 80 para el intercambio de información entre distintas unidades de control en un automóvil. Actualmente este protocolo ha despertado gran interés en otros sectores tales como la automatización industrial y el área de control. Se utiliza además como base para arquitecturas de bus industrial en aplicaciones de tiempo real distribuidas, sistemas de supervisión continua y control en el ámbito de la producción. CAN es un protocolo abierto para uso industrial y concebido como un protocolo de alta seguridad y multiplexación.

El termino CAN proviene del acrónimo inglés Controller Area Network. Es un protocolo normalizado, lo que simplifica y economiza la tarea de comunicar subsistemas de diferentes fabricantes sobre una red común o bus. El procesador delega la carga de comunicaciones a un periférico inteligente, de esta manera el procesador dispone de más tiempo para ejecutar sus propias tareas. Al tratarse de una red multiplexada, se reduce el cableado y las conexiones punto a punto.

La ISO (International Standardisation Organisation) define dos tipos de redes CAN en el automóvil:

- Una red alta velocidad, bajo el estándar ISO 11898-2, destinada para controlar el motor e interconectar las unidades de control electrónico; y
- Una red de baja velocidad, bajo el estándar ISO 11519-2 / ISO 11898-3 dedicada a la comunicación de los dispositivos electrónicos internos de un automóvil (puertas, ventanas, luces, etc).

La SAE (Society of Automotive Engineers) desarrollo sus propios estándares para su aplicación en vehículos comerciales.

Además la ISO establece dos especificaciones que están enfocadas a los requisitos de fabricación de controladores CAN, que son:

- CAN 2.0A.
- CAN 2.0B.

La diferencia más relevante entre estas dos especificaciones es que la primera define un estándar de 11 bits del identificador y la segunda especifica la trama extendida con 29 bits en el identificador.

De acuerdo al modelo de referencia OSI (Open Systems Interconnection), la arquitectura de protocolos CAN incluye tres capas: física, enlace y aplicación, además a estas se le añade una "capa especial" para la supervisión del nodo.

3.1.1. Capa física.

La capa física define los aspectos del medio físico para la transmisión de datos entre nodos de una red CAN. Los más importantes son los niveles de señal, la representación, la sincronización y tiempos en los que los bits se transfieren al bus. En el origen de las especificaciones del protocolo CAN no fue definida una capa física, sin embargo los estándares ISO 11898 e ISO11519 establecen las características que deben cumplir las aplicaciones para la transferencia en alta y baja velocidad. Esta velocidad (tiempo de bit) depende de la longitud del bus, estando las típicas entre 125 kBit/s y 1MBit/s. A continuación se ofrece una tabla que relaciona la velocidad de transferencia del bus y la longitud de los cables:

Velocidad	Tiempo de Bit	Longitud Máxima
1 Mbps	1 µS	30 m
800 Kbps	1,25 µS	50 m
500 Kbps	2 µS	100 m
250 Kbps	4 µS	250 m
125 Kbps	8 µS	500 m
50 Kbps	20 µS	1000 m
20 Kbps	50 µS	2500 m
10 Kbps	100 µS	5000 m

Tabla 3.1: Velocidad-distancia CAN.

En el cable CAN_L los valores de tensión varían entre 0 V y 2.25 V. En el CAN_H lo hacen entre 2.75 V y 5 V. En caso de interrupción de uno de los cables o derivación a masa, el sistema trabajará con el otro cable con respecto a masa.

Los módulos conectados la bus interpretan dos niveles lógicos empleando la tensión diferencial (CAN_H-CAN_L):

- ***Dominante:*** La tensión diferencial (CAN_H-CAN_L) es del orden de 2 V, con CAN_H = 3,5 V y CAN_L = 1,5 V. Sin embargo el nivel lógico interpretado es '0'.
- ***Recesivo:*** La tensión diferencial es del orden de 0 V, con CAN_H = CAN_L = 3,5 V. En este caso el nivel lógico es '1'.

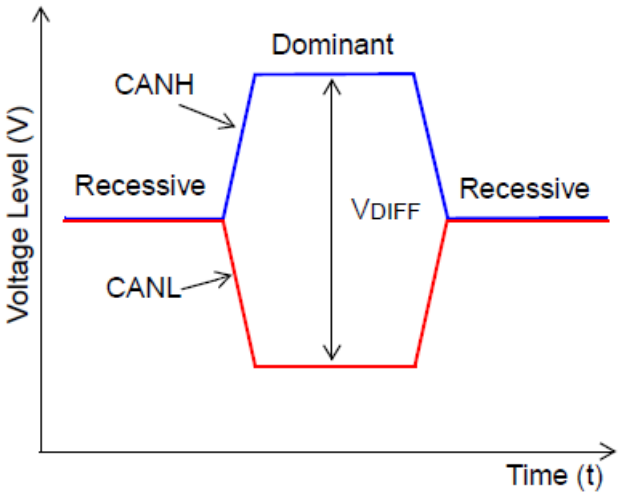


Figura 3.1: Señal diferencial en el bus CAN.

Se debe tener cuidado, ya que esta situación puede llevar a error ya que bit de CAN es dominante o recesivo y un bit de "datos" es '1' o '0'.

El par de cables trenzados CAN_H y CAN_L constituyen una transmisión de línea. Si dicha transmisión de línea no esta configurada, adecuadamente, cada trama transferida causa una reflexión que puede originar fallos de comunicación. Como la comunicación en el CAN bus, fluye en ambos sentidos, los dos extremos deben estar cerrados mediante una resistencia de 120 Ω.

3.1.2. Capa de enlace.

La capa de enlace define las tareas independientes del método de acceso al medio; por otra parte, debido a que una red CAN brinda soporte para el procesamiento en tiempo real a todos los sistemas que la integran, el intercambio de mensajes que demanda dicho procesamiento requiere de un sistema de transmisión a frecuencias alta y retrasos mínimos. En redes multimaestro, la técnica de acceso al medio es muy importante ya que todo nodo activo tiene los

derechos para controlar la red y acaparar los recursos. Por lo tanto la capa de enlace de datos define el método de acceso al medio (MAC Medium Access Control) así como los tipos de tramas para el envío de mensajes y notificaciones de sobrecarga (LLC Logical Link Control).

- ***Subnivel MAC:*** Una de las características que distingue a CAN con respecto a otras normas, es su técnica de acceso al medio denominada como CSAMA/CD+CR o "Carrier Sense, Multiple Accesss/Collision Detection + Collision Resolution" (Detección de portadora, Acceso Múltiple con Detección de Colisión + Resolución de Colisión).

  La sustitución del cableado convencional por un sistema de bus serie presenta el problema de que un nodo defectuoso puede bloquear el funcionamiento del sistema completo. Para eliminar este riesgo el protocolo CAN define un mecanismo autónomo para detectar y desconectar un nodo defectuoso del bus, dicho mecanismo se conoce como aislamiento de fallos. Cada nodo activo transmite una bandera de error cuando detecta algún tipo de error llegando a desactivarse para evitar que un nodo defectuoso pueda acaparar el medio físico.

  Gestiona, además, el tramado y desentramado de mensajes, el arbitraje a la hora de acceder al bus, reconocimiento de mensajes, de errores y su señalización, y la identificación del estado libre del bus.

- ***Subnivel LLC:*** Gestiona el filtrado de los mensajes, las notificaciones de sobrecarga y la administración de la recuperación.

3.1.3. Capa de aplicación.

Existen diferentes estándares que definen la capa de aplicación; algunos son muy específicos y están relacionados con sus campos de aplicación, como CANoe o protocolos de diseño propio.

A continuación se ofrece un esquema sistematizado de estos elementos:

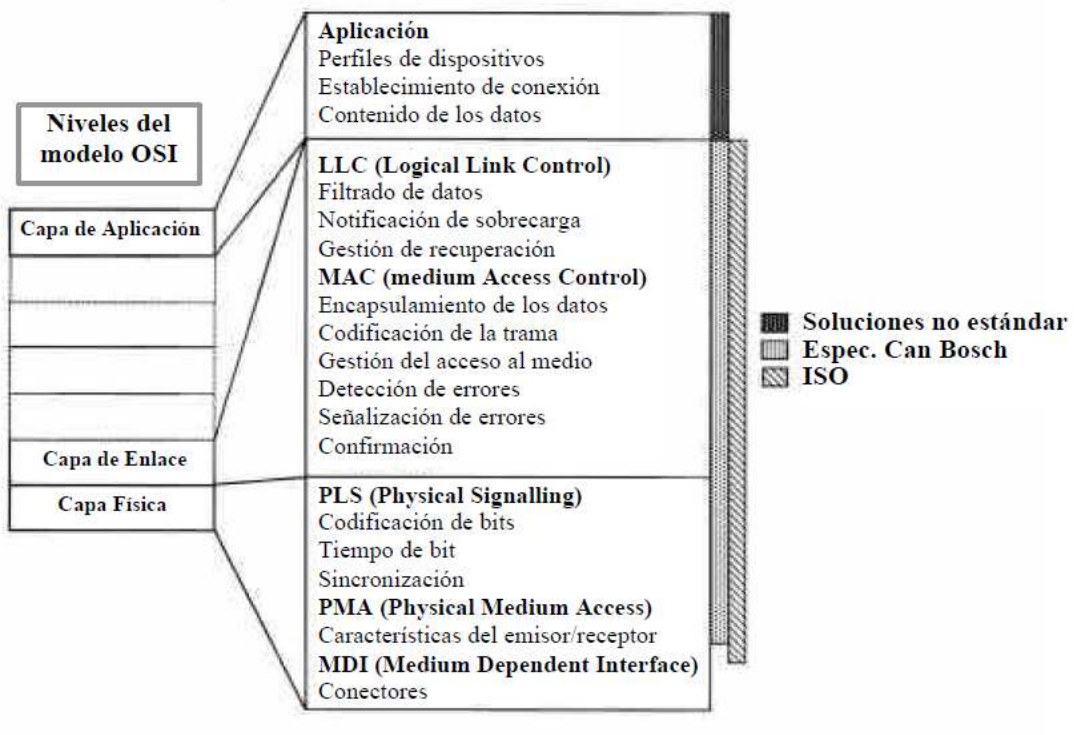


Figura 3.2: Niveles del protocolo bus CAN.

3.2. Estructura del bus CAN.

El CAN se basa en el modelo productor/consumidor, el cual es un paradigma de las comunicaciones, que se explica como un productor y uno o mas consumidores. Es un protocolo orientado a mensajes, es decir, la información del productor que se va a distribuir en mensajes, a los cuales se les asigna un identificador y se encapsulan en tramas para su transmisión, cada mensaje tiene un identificador único dentro de la red, con el cual los nodos consumidores deciden aceptar o no dicho mensaje. Un ejemplo sencillo para comprender este funcionamiento sería el caso de una conferencia telefónica, en el cual el por ejemplo un interlocutor (productor) da una charla y el resto de interlocutores escuchan lo que dice (consumidor), prestando atención o no si el tema le interesa (identificador).

La información que se encuentra en el bus se presenta en paquetes de bits con una longitud limitada y con una estructura definida de campos. Uno de estos campos es el identificador de tipo de dato del mensaje mediante el cual se sabe, la unidad que lo transmite y la prioridad respecto a otros mensajes. El mensaje entonces no va direccionado a un nodo, sino que se envía a todos los nodos y es mediante el identificador como un nodo reconoce si el mensaje le interesa o no. Todos los nodos de la red pueden ser transmisores y receptores, un nodo podrá introducir mensajes en el bus con la condición de que esté libre; si otro nodo lo intenta al mismo tiempo,

el conflicto se resuelve por la prioridad del mensaje, que va indicada por el identificador del mismo. Además el número de nodos conectados a la red es variable (dentro de unos límites). Cuando un mensaje presenta un error, el mensaje es anulado y vuelto a transmitir de forma correcta; del mismo modo un módulo con problemas avisa al resto y, si la situación no se soluciona, dicho módulo queda fuera del servicio siguiendo funcionando el resto de la red.

Las principales características de CAN son:

- Prioridad de mensajes.
- Garantía de tiempos de latencia.
- Flexibilidad en la configuración.
- Recepción por multidifusión con sincronización de tiempos.
- Sistema robusto en cuanto a consistencia de datos.
- Sistema multimaestro.
- Detección y señalización de errores.
- Retransmisión automática de tramas erróneas.
- Distinción entre errores temporales y fallas permanentes de los nodos de la red y desconexión autónoma de nodos defectuosos.

La topología en línea del bus CAN permite conectar fácilmente nuevos nodos al bus, en el caso de que un nodo esté saturando la red o introduciendo errores en el bus este se desconecta del bus, permitiendo al resto de la red funcionar con normalidad. Además, si el par de cables que conforma físicamente el bus se rompe, habrá parte del bus que podrá seguir funcionando con normalidad.

El bus CAN consta básicamente de tres partes:

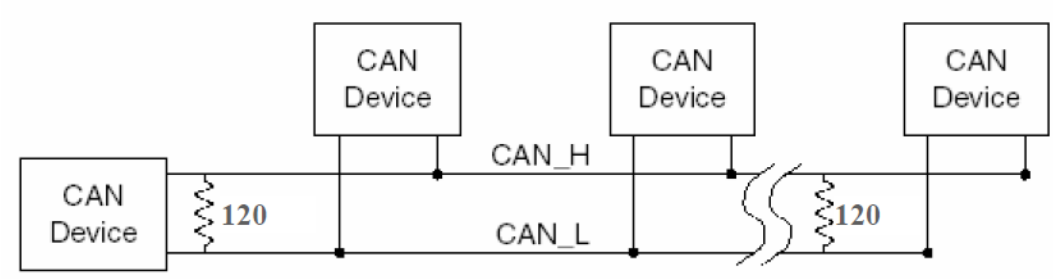


Figura 3.3: Conexión al bus CAN.

- ***Línea del bus:*** Se compone de un par de alambres trenzado al que todas las estaciones están conectadas mediante ramales cortos. En caso de que se interrumpa la línea H o que se derive a masa, el sistema trabajará con la señal L con respecto a masa, y en caso de que se interrumpa la línea L ocurrirá al contrario; de esta forma se consigue que el sistema siga trabajando con uno de los cables cortado. Es importante tener en cuenta que el trenzado sirve para anular campos magnéticos.
- ***Resistencia de carga:*** Ambos extremos de la línea del bus deben ser dotados, cada uno con su propia resistencia de carga. De esta manera aumenta la calidad de la señal, lo que permite velocidades de transmisión superiores. Sus valores son habitualmente 120 Ohmios, aunque pueden variar los valores ya que estos se obtienen para adecuar el funcionamiento del sistema a diferentes longitudes de cables y a la cantidad de módulos conectados.
- ***Estaciones:*** Las estaciones están individualmente conectadas al bus principal mediante un ramal. Cada estación supervisa los datos del bus y revisa cual es relevante y cual no de forma independiente.

  Para poder ser conectadas al bus CAN, a su vez las estaciones constan de los siguientes cuatro elementos:

  - ***Transciver:*** Que se conecta directamente a las líneas del bus y asegura que el voltaje especificado se mantiene en la línea del bus. Además protege a la estación CAN de los picos de tensión.
  - ***Controlador CAN:*** Este se conecta al transciver y recibe de él las señales del bus. Se filtran los datos irrelevantes y envía solo los datos relevantes para su procesamiento. Además, asegura que el protocolo CAN se lleva a cabo durante la transmisión de datos y es en este punto donde se detectan los errores de transmisión reaccionando de forma apropiada en cada caso.
  - ***Buffer:*** Se trata de la memoria asignada para las colas de recepción y envío de mensajes.
  - ***Microcontrolador:*** Cuenta con un programa para el procesado y la utilización de los datos que se reciben desde el controlador CAN. Se puede preguntar por el estado de los sensores conectados y controlar los actuadores conectados. El tipo de sensores y actuadores dependen del tipo de aplicación.

En la práctica estos cuatro componentes se colocan total o parcialmente en un único microchip, ahorrando espacio y costes.

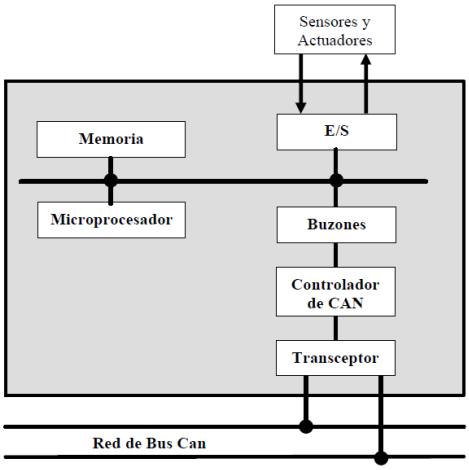


Figura 3.4: Ejemplo estación o nodo CAN.

La estructura de los buzones permite diferenciar dos tipos de implementaciones, Basic CAN y Full CAN. La diferencia esta en como los mensajes recibidos son filtrados usando el identificador de mensaje antes de ser escritos en el buzón correspondiente, de esta manera:

- ***Basic CAN:*** Interrumpe al microcontrolador constantemente con cada mensaje recibido y enviará cada mensaje bajo alguna orden.
- ***Full CAN:*** En cambio este tipo tiene dispositivos extra que le permiten recibir y transmitir mensajes automáticamente, sin generar tantas interrupciones en el microcontrolador.

3.2.1. Desarrollo de la transmisión.

El protocolo CAN utiliza una comunicación serie asíncrona. Los módulos que se conectan al sistema CAN bus son los que necesitan compartir información, pertenezcan o no a un mismo sistema. Tratándose de coches es razonable que la velocidad de transmisión de los datos del ABS deba de ser muy rápida por la importancia de la información y, sin embargo, la del climatizador no hace falta que disponga de una línea tan rápida ya que el muestreo de estos datos no es tan importante.

Como ya se ha dicho, el sistema CAN está orientado hacia el mensaje y no hacia el destinatario. La información en la línea es transmitida en forma de mensajes estructurados, de forma que una

parte del mismo es un identificador que indica la clase de dato que contiene. Todos los módulos reciben el mensaje, lo filtran y solo lo emplean los que le necesitan. Como se supone que todos los módulos conectados al sistema son capaces tanto de introducir como de recibir mensajes de la línea. Cuando el bus esta libre cualquier módulo conectado puede empezar una nueva transmisión.

En el caso de que uno o varios módulos quieran introducir un mensaje al mismo tiempo, lo hará el que tenga una mayor prioridad. Esta prioridad nos la indica el identificador del mensaje. De tal forma que si inicialmente dos módulos están siendo transmisores, ambos comienzan mandando el identificador del mensaje, mientras no haya colisión los dos siguen siendo transmisores, pero cuando uno de ellos este transmitiendo un '1' y detecta que otro módulo está transmitiendo un '0' el que trasmitía el '1' deja de ser transmisor y pasa a escucha este arbitraje se realiza mediante la técnica CSAMA/CD+CR de la capa de aplicación.

En un bus, un identificador de mensaje ha de ser asignado a un solo módulo concreto, evitándose que dos módulos puedan iniciar la transmisión a la vez con un mismo identificador y mensajes diferentes. En CAN la filosofía es que cada mensaje es único en el bus.

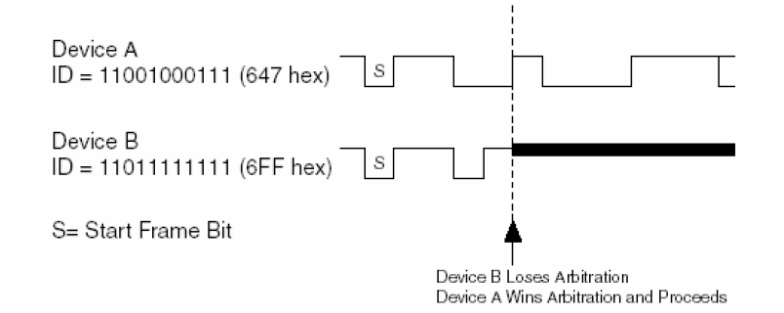


Figura 3.5: Arbitraje de prioridad de envío CAN.

El proceso de transmisión de datos se realiza en 5 pasos:

1. Un módulo recibe información de los sensores que tiene asociados o del propio bus CAN (rpm de un motor, velocidad, temperatura del motor, puerta abierta, etc.). Su microcontrolador la procesa y pasa la información al controlador donde es gestionada y preparada para ser traducida a señales eléctricas compatibles con el estándar CAN.

2. El controlador de dicho módulo transfiere los datos y el identificador junto con la petición de inicio de transmisión, asumiendo la responsabilidad de que el mensaje sea

correctamente trasmitido a todos los módulos de la red. Para transmitir el mensaje el controlador ha tenido que encontrar el bus libre y en caso de conflicto con otro módulo intentando transmitir al mismo tiempo tener una prioridad mayor. A partir del momento en que esto ocurre el resto de módulos se convierten en receptores.

3. Todos los módulos conectados al bus reciben el mensaje.

4. Verifican el identificador para determinar si el mensaje va a ser utilizado por ellos.

5. Los módulos que necesiten los datos del mensaje lo procesan, sino lo necesita, el mensaje se ignora.

El sistema CAN bus dispone de mecanismos para detectar errores en la transmisión de mensajes, de forma que todos los receptores realizan un chequeo del mensaje analizando una parte del mismo, el campo llamado CRC. Otros mecanismos de control se aplican en los emisores que mirarán el nivel del bus, la presencia de campos de formato fijo en el mensaje (verificación de la trama), análisis estadísticos por parte de los módulos de sus propios fallos, etc. Estas medias hacen que las probabilidades de error en la emisión y recepción de mensajes sean muy bajas, por lo que es un sistema muy seguro.

3.2.2. Detección de errores.

En el momento en que un módulo detecta un error en una trama este dispositivo transmite una secuencia especial de bits llamada “trama de error” (apartado 3.3.3). Cuando el dispositivo que ha transmitido la trama errónea detecta esta tarama, transmite la trama de nuevo.

Los dispositivos de CAN detectan los siguientes errores:

- ***Bit Error:*** Durante la transmisión de una trama, el emisor a la vez monitoriza el bus. Cualquier bit que reciba con polaridad inversa a la que ha transmitido se considera un error de bit, exceptuando cuando se está recibiendo el campo de arbitraje. Tampoco es bit de error la detección de bit dominante por un nodo en estado de error pasivo que transmite una trama de error pasivo.
- ***Stuff Error:*** Error de relleno es la detección de 6 bits consecutivos del mismo nivel en cualquier campo que siga la técnica de relleno de bits, donde por cada 5 bits iguales se añade uno diferente.

- ***CRC Error:*** Cuando el cálculo del receptor no coincide con el recibido en la trama. El campo CRC (Cyclic Redunant Code) contiene 15 bits. Esta medida sirve para detectar errores de transmisión debido a problemas en el medio físico como, por ejemplo, el ruido.
- ***Form Error:*** Error de forma, detecta un bit dominante cuando en un campo de formato fijo se recibe alterado algún bit, estos son: final de trama, espacio entre ramas, delimitador ACK y delimitador CRC.
- ***Acknowledgment Error:*** El error de reconocimiento se produce cuando ningún nodo cambia a dominante el bit de reconocimiento.

El protocolo CAN detecta diversos fallos en la línea física de comunicación como línea desconectada o cortocircuitadas, aunque no especifica cómo reaccionar en caso que se produzca alguno de estos errores.

3.2.3. Aislamiento de módulo defectuoso.

Para evitar que un nodo con problemas condicione el funcionamiento del bus, a la especificación CAN se le han incorporado unas medidas para el aislamiento de nodos defectuosos que son gestionadas por los controladores. Un nodo puede estar en uno de los tres estados siguientes en relación con la gestión de errores:

- ***Active Error:*** Error activo es el estado normal de un nodo. Este puede participar en las comunicaciones y en caso de detección de error envía una trama de error activa.
- ***Passive Error:*** Un nodo en estado de error pasivo participa en la comunicación, sin embargo ha de esperar una secuencia adicional de bits recesivos antes de intentar transmitir. Solo puede señalar errores con una trama de error pasivo.
- ***Bus off:*** Nodo anulado. En este estado se deshabilita el controlador CAN y no participará en la comunicación

La transición entre los distintos estados del nodo se basa en dos contadores incluidos en el controlador: Contador de errores de transmisión (TEC) y contador de errores de recepción (REC). Cuando alguno de estos supera 127 se produce la transición a estado pasivo y cuando TEC es mayor de 255 el nodo pasa a estado de Bus off. Además estos contadores permiten al nodo internamente detectar dónde está el fallo.

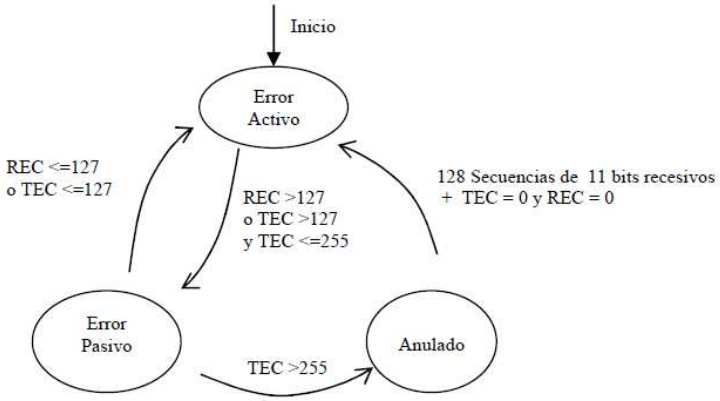


Figura 3.6: Transición de estados de error CAN.

El planteamiento del CAN bus, como puede observarse, permite disminuir notablemente el cableado, puesto que si un módulo dispone de una información, por ejemplo, la temperatura ambiente, ésta puede ser utilizada por el resto de módulos sin que sea necesario que cada uno de ellos reciba la información de ese sensor de temperatura. Otra ventaja palpable es que las funciones pueden ser repartidas entre los distintos módulos, de forma que si se necesitan más funciones no suponga un problema económico importante, simplemente bastaría con ampliar la red.

3.3. Tipos y formato de Tramas.

El mensaje es una sucesión de “0” y “1”, que están representados por los distintos niveles de tensión en los cables del CAN bus. CAN utiliza mensajes de estructura predefinida para la gestión de la comunicación, llamados tramas.

En un bus CAN los módulos transmiten la información, con tramas de datos, sin necesidad de una orden, bien sea por un proceso realizado con una frecuencia o bien sea activado ante algún suceso en el módulo. La trama de interrogación remota solo se suele utilizar para detectar la presencia de módulos o para la puesta al día de información de un módulo recién conectado a la red. Los mensajes pueden entrar en colisión, pero como ya se ha explicado el de identificador de mayor prioridad prevalecerá y los demás se transmitirán posteriormente, retransmitidos lo antes posible.

Se distinguen dos variantes del CAN, el definido en CAN 2.0 A o “CAN Standard (Basic CAN)” y el definido en CAN 2.0 B o “CAN Extendido (PeliCAN)”, los formatos de las tramas son idénticos, como se verá más adelante. La diferencia más sustancial es que el número de bits que se emplean para el identificador de mensaje: 11 bits (2032 identificadores) diferentes en CAN Standard y 29 bits (536870912 identificadores) en CAN Extendido. Las tramas de CAN

son de longitud reducida, la trama más larga es de 130 bits en CAN estándar y 154 bits en CAN extendido.

Las tramas se descomponen en campos de diferente tamaño (número de bits) que permiten llevar a cabo el proceso de comunicación entre los módulos según el protocolo definido por Bosch para CAN bus, facilitando desde identificar al módulo hasta indicar el principio y el final del mensaje, mostrar los datos, permitir distintos controles, etc.

Los tipos de trama y estados del bus utilizados son:

3.3.1. Trama de datos.

La trama de datos es la que un módulo utiliza normalmente para poner información en el bus. Puede incluir entre 0 y 8 Bytes de información útil. Como ya se ha dicho existen dos tipos de tramas de datos, la estándar y la extendida. A continuación se detalla la trama estándar y más abajo se detalla la trama extendida.

3.3.1.1. Trama de datos estándar.

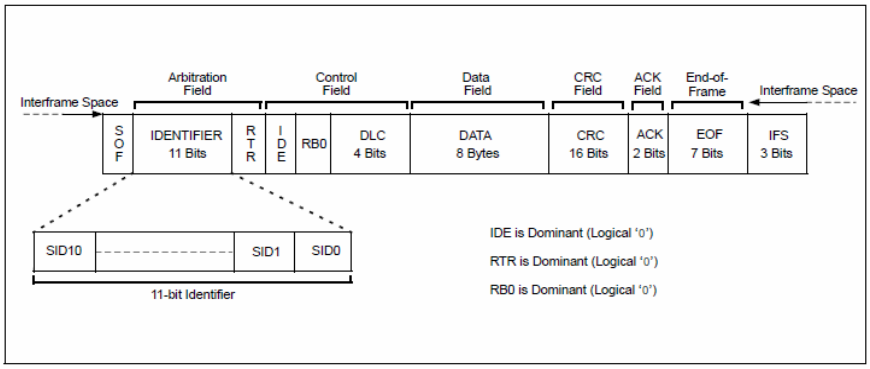


Figura 3.7: Trama de datos estándar.

Las tramas de datos estándar comienzan con un bit de Start-of-Frame (SOF), seguido del campo Arbitration de 12 bits, tal como se muestra en la figura. El campo Arbitration se compone de un identificador de 11 bits y un bit de Petición de Transmisión Remota (RTR). El identificador define el tipo de información contenida en el mensaje y es usado por cada nodo receptor para determinar si el mensaje es de su interés o no. El bit RTR sirve para distinguir una trama de datos de una trama remota. En una trama de datos estándar, el bit RTR está inactivo (estado lógico '0').

Después del campo Arbitration está el campo Control, de 6 bits, el cual proporciona más información acerca del contenido del mensaje. El primer bit del campo Control, es el bit IDE (Identifier Extension), que ayuda a distinguir entre tramas de datos estándar y extendidas. Una trama de datos estándar se indica mediante un estado dominante (nivel lógico '0') durante la transmisión del bit IDE. El segundo bit del campo Control es un bit reservado (RB0), el cual está a nivel bajo. Los últimos cuatro bits del campo Control representan el DLC (Data Length Code) o tamaño del campo de datos, que especifica el número de bytes de datos que contiene el mensaje.

El siguiente campo es el campo de datos. Este campo contiene los datos del mensaje. El tamaño de este campo puede variar desde 0 hasta 8 bytes. El número de bytes es configurable por el usuario.

El campo de datos va seguido del campo CRC (Cyclic Redundancy Check) que es una secuencia CRC de 15 bits seguido de un bit delimitador.

El campo Acknowledgement (ACK) se compone de dos bits, el ACK SLOT y el ACK DELIMITER. El transmisor envía estos dos bits en estado recesivo (nivel lógico '1') y el receptor que recibe el mensaje correctamente sobrescribe el bit ACK SLOT con un bit a nivel bajo.

El último campo es el campo End-of-Frame (EOF), que consiste en 7 bits a nivel alto que indican el final del mensaje.

3.3.1.2. Trama de datos extendida.

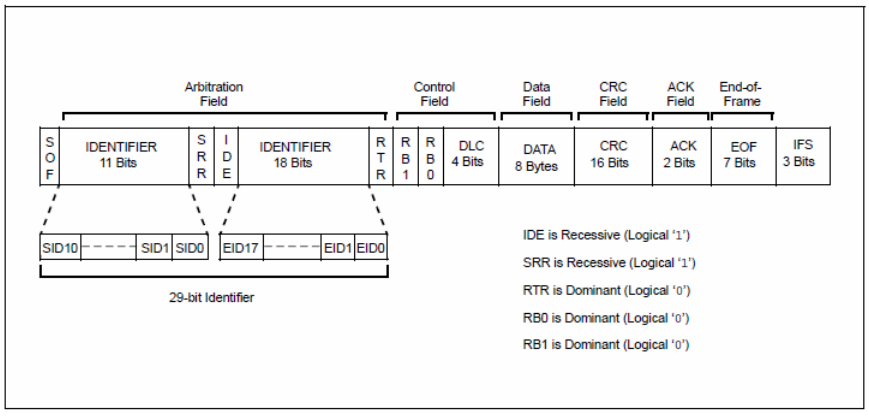


Figura 3.8: Trama de datos extendida.

La trama de datos extendida también comienza con el campo SOF y va seguido de un campo Arbitration de 31 bits, tal como se muestra en la figura .5. El campo Arbitration de las tramas de datos extendidas contiene un identificador de 29 bits separado en dos campos por los bits

SRR (Substitute Remote Request) e IDE. El bit SRR = 1 en las tramas de datos extendidas. El bit IDE indica el tipo de trama de datos. En las tramas extendidas IDE = 1.

El campo Control en las tramas de datos extendidas es de siete bits. El primer bit es el RTR, que vale 0 en este tipo de tramas. Los siguientes dos bits, RB1 y RB0, son bits reservados y se encuentran a nivel bajo. Los últimos cuatro bits del campo Control representan el DLC (Data Length Code) o tamaño del campo de datos, que especifica el número de bytes de datos que contiene el mensaje.

Los siguientes campos de las tramas de datos extendidas son idénticos a los de las tramas estándar.

3.3.2. Trama de interrogación remota.

Puede ser utilizada por un módulo para solicitar la transmisión de una trama de de datos con la información implicada a un identificador dado. El módulo que disponga de la información definida por el identificador la transmitirá en una trama de datos.

Un nodo que está esperando recibir datos de otro nodo puede iniciar la transmisión enviando al nodo transmisor una trama remota. La trama remota puede estar en formato estándar o en formato extendido.

Una trama remota es similar a una trama de datos, con las siguientes excepciones:

- El bit RTR es recesivo (RTR = 1).
- No hay campo de datos (DLC = 0).

3.3.2.1. Trama remota estándar.

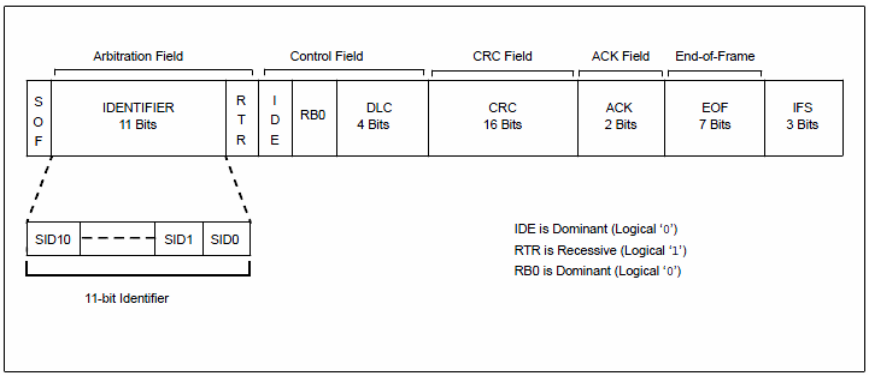


Figura 3.9: Trama remota estándar.

3.3.2.2. Trama remota extendida.

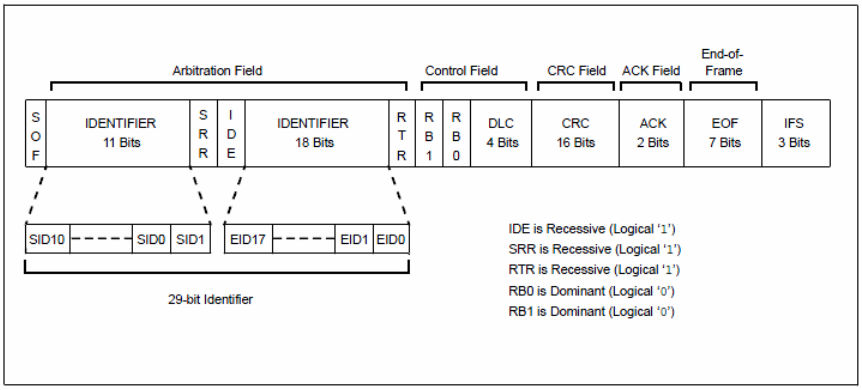


Figura 3.10: Trama remota extendida.

3.3.3. Trama de error.

Una trama de error es generada por cualquier nodo que detecte un error en el bus. Estas tramas se componen de dos campos: indicador de error (“Error Flag”) y el Delimitador de error (Error Delimiter). El campo Error Delimiter consiste en 8 bits a nivel alto y permite a los nodos del bus restablecer las comunicaciones limpiamente después de un error. Hay dos tipos de campos Error Flag, dependiendo del estado del nodo que detecta el error:

- ***Error Flag Activo:*** Si un módulo en estado de error activo detecta un error en el bus, interrumpe la comunicación del mensaje en proceso generando un indicador de error activo, que contiene 6 bits a nivel bajo consecutivos, secuencia que rompe la regla de relleno de bits lo que fuerza a otros nodos de la red a generar Error Echo Flags, resultando por tanto en una serie de 6 a 12 bits a nivel bajo en el bus. Finalmente el campo que delimita el error formado por 8 bits recesivos. Entonces la comunicación se reinicia.
- ***Error Flag Pasivo:*** Cuando un módulo en estado de error pasivo detecta un error, el módulo transmite un indicador de error pasivo contiene 6 bits a nivel alto consecutivos y, por tanto la trama de error es una secuencia de 14 bits recesivos. Con el resultado de que a menos que el error en el bus sea detectado por el nodo transmisor, la transmisión de un flag de error pasivo no afectará a las comunicaciones de ningún otro nodo de la red.

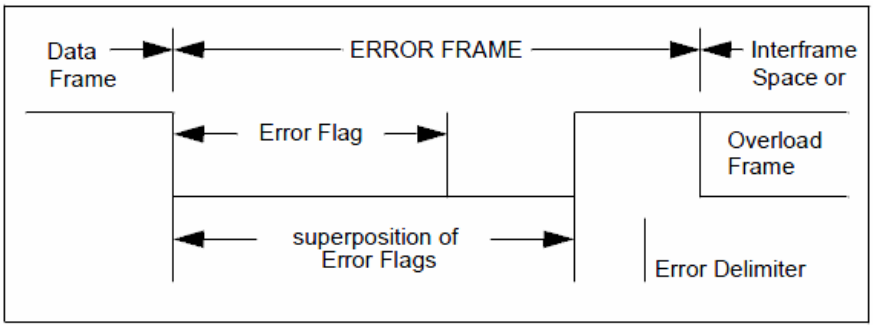


Figura 3.11: Trama de error.

3.3.4. Trama de sobrecarga.

Permite que un módulo fuerce a los demás a alargar el tiempo entre transmisión de tramas sucesivas. Lo hace el dispositivo con independencia del software.

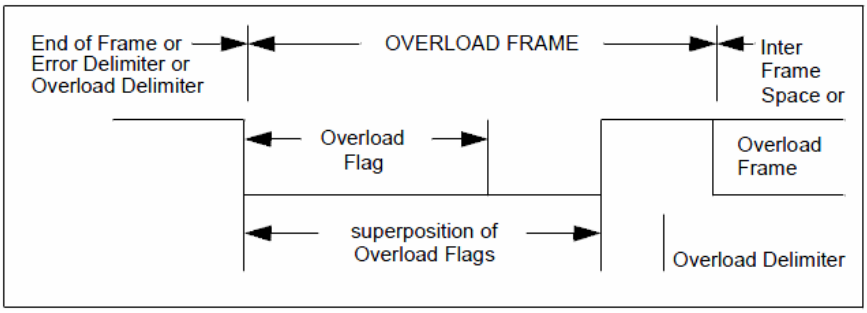


Figura 3.12: Trama de sobrecarga.

Una trama de sobrecarga puede ser generada por un nodo, tanto cuando se detecta un bit a nivel bajo durante el espacio entre tramas (Interframe Spacing), como cuando un nodo todavía no está listo para recibir el siguiente mensaje (por ejemplo, si está todavía leyendo el mensaje recibido previamente). Otra causa para iniciar la transmisión de una trama de sobrecarga es la detección por cualquier módulo de un bit dominante en los 3 bits inter-trama. De esta forma un nodo puede generar un máximo de dos tramas de sobrecarga seguidas para retrasar el comienzo de la transmisión de un nuevo mensaje.

Una trama de sobrecarga tiene el mismo formato que una trama de error con el flag de error activo, pero solo puede generarse durante el espacio entre tramas. Consiste en un campo indicador de sobrecarga (Overload Flag), compuesto por 6 bits a nivel bajo que pueden ser seguidos por las tramas de sobrecarga generadas por otros módulos dando lugar a 12 bits dominantes como máximo, seguido de un campo delimitador (Overload Delimiter), compuesto de 8 bits a nivel alto.

3.3.5. Espaciado inter-tramas.

Las tramas de datos (y de interrogación remota) se separan entre sí por una secuencia predefinida que se denomina espaciado inter-trama (IFS Inter Frame Space).

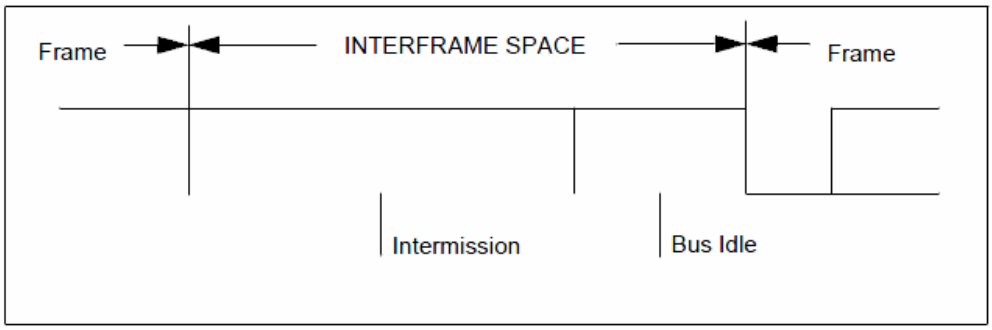


Figura 3.13: Interframe space para nodos que no están en estado de error pasivo.

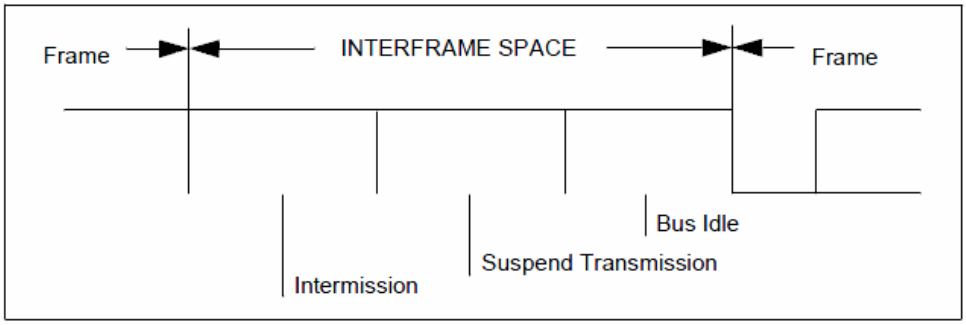


Figura 3.14: Interframe space para nodos que están en estado de error pasivo.

El Interframe Space o espacio entre tramas separa dos tramas sucesivas transmitidas al bus CAN. Consiste en al menos 3 bits a nivel alto. El espacio entre tramas proporciona tiempo a los nodos para procesar internamente el mensaje recibido previamente, antes del comienzo de la siguiente trama. Si el nodo transmisor está en el estado de error pasivo, se insertan 8 bits a nivel alto adicionales en el espacio entre tramas antes de que el nodo transmita otro mensaje. Este periodo se conoce como campo de transmisión suspendida y proporciona tiempo a otros nodos de la red para tomar el control del bus.

3.3.6. Bus en reposo.

En los intervalos de inactividad se mantiene constante el nivel recesivo del bus.

3.4. Control FIFO.

El módulo CAN utiliza la memoria RAM del microcontrolador para el almacenamiento de los mensajes CAN que deben ser transmitidos o han sido recibidos en colas FIFO. El módulo por sí mismo no tiene buffers de mensajes accesibles por el usuario. La siguiente figura muestra la organización de la memoria en un módulo CAN.

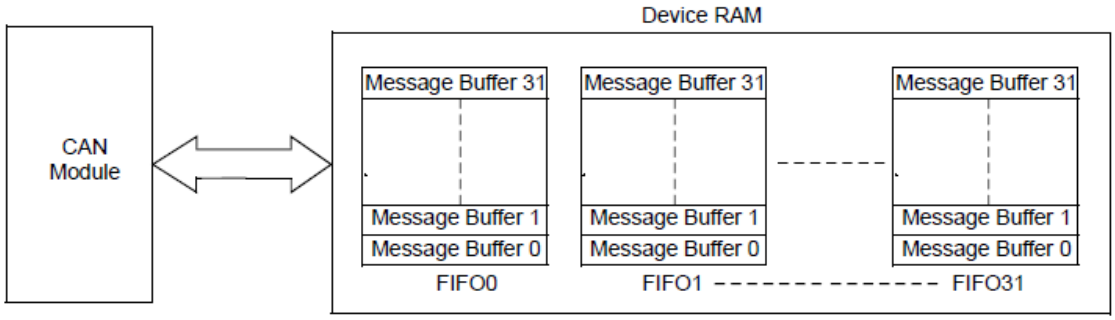


Figura 3.15: Organización de la memoria.

Como en una red CAN la difusión de un mensaje es broadcast, es decir, a todos. Un mensaje transmitido por un nodo es recibido por todos los nodos de la red. De forma individual cada nodo de la red necesita un mecanismo de filtrado para recibir los mensajes de su interés. Este mecanismo lo proporcionan los filtros de aceptación de mensaje y los registros de mascara. El filtrado se realiza con el campo identificador del mensaje.

3.4.1. FIFO Rx/Tx.

Cada buffer FIFO puede ser configurado de forma individual como buffer de transmisión (Tx) o de recepción (Rx). Para cada mensaje dentro de cada buffer se han de dejar 16 bytes de espacio.

El módulo CAN solo requiere la dirección de inicio de la primera posición de la FIFO, esta se llama dirección base de CAN FIFO. El módulo CAN calcula automáticamente la dirección de los siguientes buffers FIFO, basándose en la configuración individual de cada uno de estos buffers FIFO. La memoria de las FIFOs individuales se disponen contiguamente. Esto produce una memoria sin espacios dedicada a los mensajes de CAN.

El usuario debe fijar el tamaño de cada FIFO. Entonces la memoria total que se asigna se obtiene contando el número total de buffers FIFO y el tamaño de memoria reservado para cada buffer.

3.4.2. Filtrado y mascaras.

La aplicación de usuario configura el filtro específico para recibir un mensaje con un identificador dado mediante el establecimiento de un filtro para que coincida con el identificador del mensaje que se quiere aceptar. Cada filtro es controlado de forma individual.

Como los mensajes se reciben por el módulo CAN, el identificador de mensaje se compara con los bits correspondientes en los filtros. Si el identificador coincide con el filtro configurado por el usuario, el mensaje se almacenará en la FIFO a la que apunte la configuración del filtro, según sus registros de configuración.

Las mascaras de aceptación se puede utilizar para ignorar los bits seleccionados del identificador a medida que se reciben. Estos bits no serán comparados con los bits en los filtros cuando se recibe el mensaje. Por ejemplo, si el usuario desea recibir todos los mensajes con identificadores de 0, 1, 2 y 3, el usuario tendría que enmascarar los dos bits más bajos del identificador. Un filtro puede tener asociado una única mascara de aceptación, pero esta mascara la pueden emplear otros filtros.

4. Desarrollo hardware.

4.1. Diagrama de bloques hardware.

Para la solución hardware del proyecto se propone un trabajo empleando un sistema basado en microcontrolador, el movimiento de los "runners" se hará con un motor lineal paso a paso. Se deben adquirir las rpm en las que se encuentra el motor al igual que la posición del acelerador. Dando un esquema de trabajo análogo para hardware como para software.

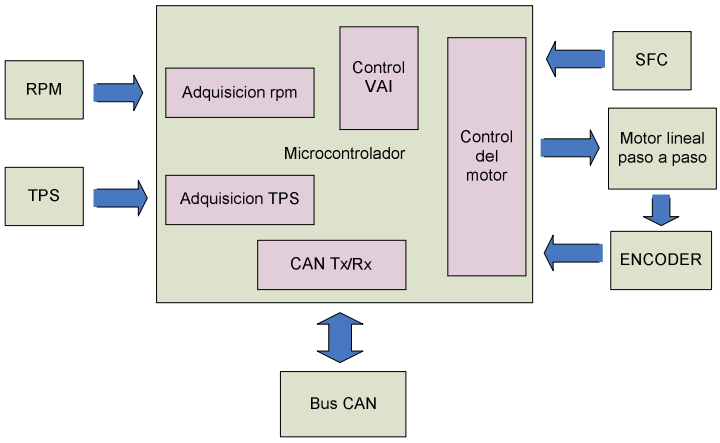


Figura 4.1: Esquema de trabajo.

En la figura 4.1 en verde se tienen las diferentes partes del hardware que son necesarias para la realización del proyecto.

- ***RPM:*** Este bloque consta de un sensor que transmitirá mediante una señal cuadrada variable en frecuencia las rpm del motor del monoplaza.
- ***TPS:*** Es un circuito acondicionador para la señal procedente del sensor TPS, la señal esta en la figura 1.8.
- ***Bus CAN:*** Componentes necesarios para conectar el microcontrolador a la red CAN.
- ***Motor lineal paso a paso:*** Aquí se encuentra el motor lineal paso a paso así como la electrónica para poder moverlo.
- ***SFC:*** Sensor de Fin de Carrera pulsadores para detectar cuando se llega al final del recorrido.
- ***ENCODER:*** Para conocer la posición real del motor paso a paso. Como no se dispone de uno se emplea la señal nHOME del DRV8805 para simularla.
- ***Microcontrolador:*** Conexionado de los distintos bloques hardware al microcontrolador junto con conexión para programación y depuración con PICkit3.

De color rosado están los apartados de trabajo software con los que los distintos bloques hardware se conectan.

- ***Control VAI:*** Control Variable Air Intake, es un programa que se encarga del gobierno del sistema controlando como se actúa sobre el motor, gestiona las tablas de posicionamiento y los mensajes enviados y recibidos.
- ***Adquisición rpm:*** Programación del IC (Input Capture) para la lectura del sensor de rpm del motor de combustión.
- ***Adquisición TPS:*** Programación del ADC para la conversión de la señal del TPS.
- ***CAN Tx/Rx:*** Configuración de la comunicación CAN, gestión de errores y estado del bus.
- ***Control motor:*** Se disponen de varios periféricos del micro para el control del movimiento del motor paso a paso, el puerto D, la interrupción externa 3 que actúa como SFC (Sensor de Fin de Carrera), OC (Output Compare) y el timer 4.

4.2. Alimentación.

El bloque de alimentación se encarga de proporcionar tensiones estables a todos los elementos del sistema, a partir de los 12 V de la batería de un coche. Para su diseño se analizaron las distintas necesidades de tensión en el circuito para su funcionamiento. Se va ha necesitar una tensión de 3.3 V, para el microcontrolador, 5 V requiere el MCP2551 y es necesaria una tensión simétrica para alimentar los amplificadores del circuito acondicionador de TPS, esta tensión debe ser al menos +/-6.5 V. Se opta por una conversor DC-DC el NMA0512SC de baja potencia para no encarecer el coste del circuito y asegura unos niveles de tensión apropiados y sin grandes variaciones, además se emplean dos reguladores, el LM7805 y el regulador de bajo drop-out LM3940.

También se realiza una conexión especial de alimentación hacia el DRV8805, para permitir si se desea cambiar la alimentación del motor. Ese bloque son condensadores para estabilizar la tensión.

4.2.1. Conversor NMA0512SC.

Es un conversor DC-DC de tensión, al conectarse 5 V en su entrada genera en sus salidas unas alimentación simétrica de +/-12 V respecto a un común 0 V. Este común se debe conectar a la masa del circuito para que esta fuente de alimentación este referenciada al mismo punto que el

resto del circuito. Aun que solo se tienen 42 mA, estos son mas que suficientes para alimentar la parte de circuito que necesita de estos niveles de tensión simétricos.

4.2.2. Esquemático de la conexión.

Así la conexión entre los elementos se hace, primero de la batería del coche al LM7805 y de aquí al LM3940 y al NMA0512SC, esta configuración puede cambiar, pero situar primero el conversor encarece el circuito, ya que se ha de buscar un conversor capaz de suministrar mas de 160 mA que es la demanda de corriente del circuito con dicha configuración.

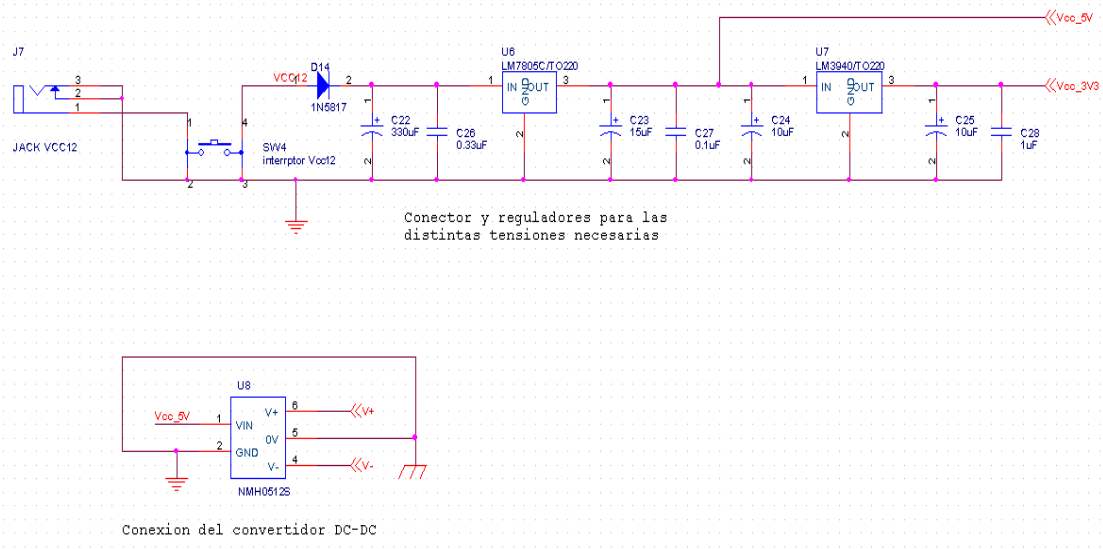


Figura 4.2: Esquemático de la fuente de alimentación.

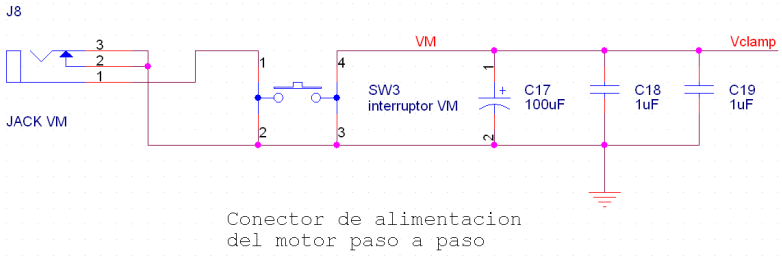


Figura 4.3: Esquemático de la alimentación para el motor paso a paso.

4.3. El microcontrolador PIC32MX795f512-L.

La elección del fabricante del microcontrolador se basa en la disponibilidad de diverso material en el laboratorio para el desarrollo del proyecto. La elección de la familia PIC32MXxxx se basa en el uso de un microcontrolador con arquitectura de 32 bits, tener una frecuencia de trabajo

hasta 80 MHz y la disponibilidad de buses separados de instrucción y datos que le dan mayor velocidad de procesado. Además de la realización de multiplicaciones en un único ciclo, MDM (Multiple Divide Unit) y dos sets de 32 registros en el núcleo para reducir tiempos de latencia hacia interrupción y viceversa.

La memora disponible en la familia esta organizada en dos, una memoria no volátil Flash, de 64K a 512K más una memoria de arranque de 12KB y una memoria volátil SRAM de 16K hasta 128K.

Una particularidad son sus múltiples vectores de interrupción con prioridad independiente programable.

Es capaz de realizar operaciones sin ser interrumpidas sobre los registros de periféricos, los periféricos de los que dispone son:

- 8 canales hardware DMA (Direct Memory Access) con detección automática del tamaño del dato.
- USB 2.0 y controlador OTG (On The Go), con un canal DMA dedicado.
- 10/100 Mbps Ethernet MAC con interface MII y RMII. Y canales DMA dedicados.
- Módulo CAN 2.0B, con soporte para direccionamiento con DeviceNet y canales DMA dedicados.
- 6 módulos UART con soporte para RS-232, RS-485 y LIN.
- 4 módulos SPI.
- 5 módulos I2C.
- Puerto paralelo maestro y esclavo con capacidad de direccionar hasta 16 bits.
- RTCC.
- 5 timer/counter 16 bit. Combinándolos en parejas se consiguen dos timer/counter de 32 bit.
- 5 Capture inputs.
- 5 Compare/PWM outputs.
- 5 interrupciones externas.
- 16 canales ADC 10 bits, con una tasa de conversión de 1Msps.
- 2 comparadores analógicos.
- Puertos son de alta velocidad capaces de conmutar a velocidad de 80 MHz.

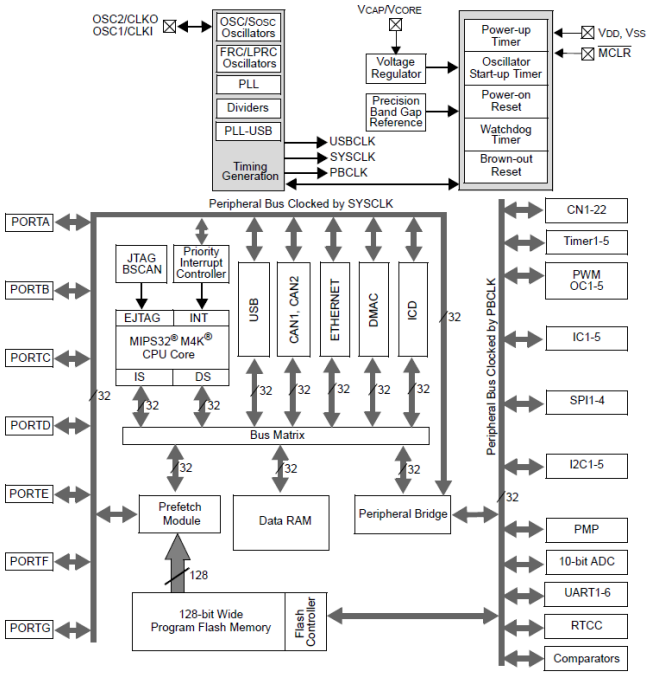


Figura 4.4: Diagrama de bloques de PIC32MX.

4.3.1. Circuito reset/depuración y desacoplo.

Es necesario el uso de condensadores de desacoplo en los pines Vdd, Vss, AVdd y AVss. De modo que en el diseño de la PCB se ha de prestar atención a situar y conectar adecuadamente estos condensadores de desacoplo. Es necesario también otro condensador en el pin Vcap/Vcore para estabilizar la tensión de referencia interna.

Los pines /MCLR, PGEC2 y PGED2 son los que se van a emplear para la programación y depuración, siguiendo las recomendaciones que se hacen en el datasheet para la elección de los componentes que conforman el circuito.

4.3.2. Esquemático de la conexión.

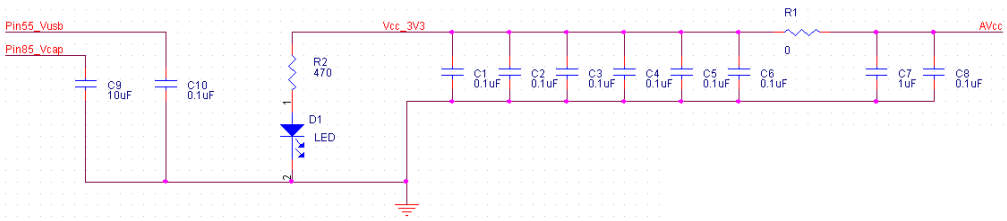


Figura 4.5: Esquemático de la alimentación del PIC32MX.

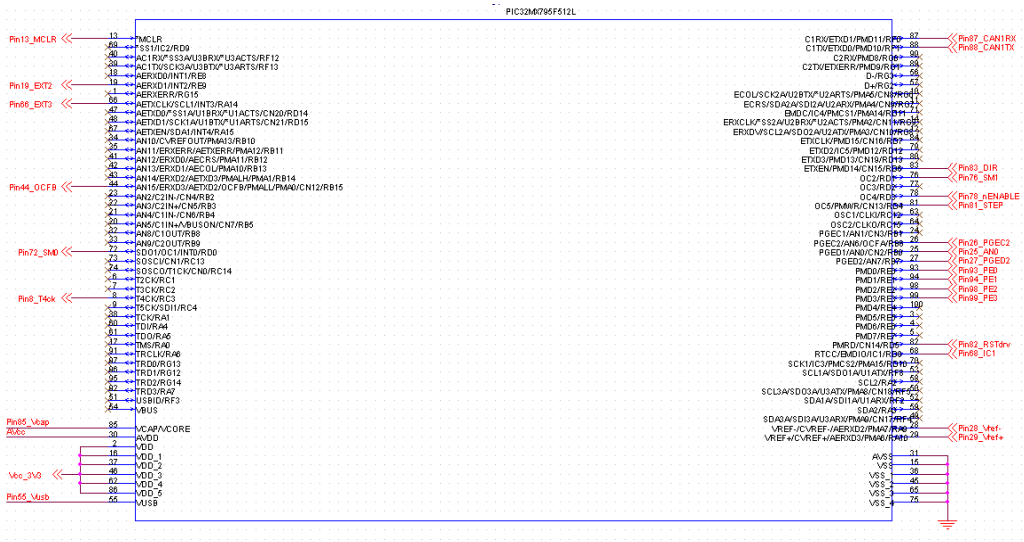


Figura 4.6: Esquemático de conexión del PIC32MX795F512L.

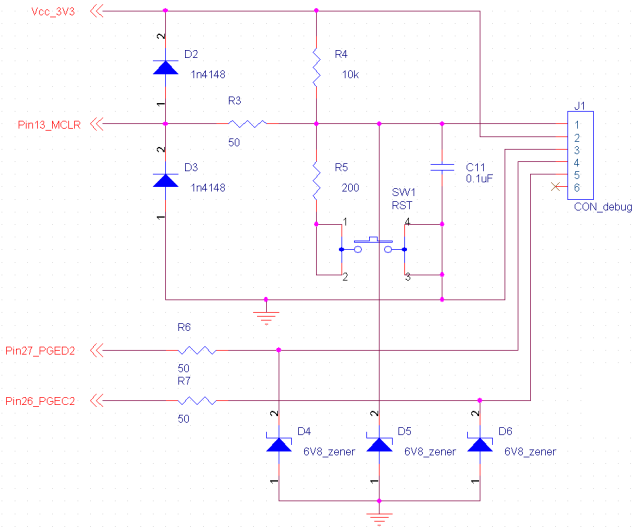


Figura 4.7: Esquemático del circuito depurador/programador y reset.

4.4. Selección del Actuador.

La elección de un motor lineal paso a paso se hace a partir de las necesidades del movimiento a realizar, para el diseño del movimiento del paso a paso del sistema VAI se emplea la información aportada por el INSIA en el capítulo 1. El motor eléctrico necesitará realizar la mayor aceleración cuando la distancia a recorrer por unidad de tiempo sea menor. Como el desplazamiento del motor se realiza en escalones, el motor necesitará acelerar más rápido cuanto más pequeño sea este escalón para respetar una velocidad de movimiento del motor solicitada de 200mm/s. Ya que no se indica el número de posiciones minima, se va a realizar el diseño para que se pueda poner un escalón de distancia por cada incremento de 100 rpm en el

motor de combustión. De la información del INSIA se extrae que el motor de combustión tarda 14.5 ms en realizar este incremento de rpm, se espera que el motor al menos sea capaz de desplazarse 3 mm en este. Pudiéndose disponer de un total de hasta 70 posiciones distintas donde situar los "runners". Se desea fijar una resolución mínima de posición de 1 mm. ya que lo único que se sabe es que un recorrido de 180 mm ha de realizarse en 1 segundo como máximo, entonces 1 mm se debe recorrer en 5.5 ms como máximo, para asegurar que el motor se mueve en tramos pequeños de distancia a la velocidad adecuada.

También se debe saber la inercia del la masa sobre la que le motor tiene que actuar. La inercia es la tendencia de los cuerpos a mantener el estado en el que se encuentran, ya que se trata de un peso de 0.4 kg desplazándose en el eje vertical la fuerza de inercia de los "runners" es de 3.92 N. El cálculo de esta fuerza se obtiene de la siguiente ecuación.

$$T_L(f) = m * g$$

Ecuaciones 4.1: Fuerza de inercia.

Ahora se ha de buscar un motor capaz de desplazarse a 200 mm/s, un empuje mayor de 3,92 N. Será un buen criterio que el tiempo que tarde en alcanzar dicha velocidad de trabajo sea lo más baja posible y así tener un buen tiempo de respuesta en el posicionado de los "runners". Siendo interesante, si el empuje lo permite, hacer que esta fuerza de inercia sea la del conjunto motor + "runners" dejando mas posibilidades en el momento de situar el paso a paso en el monoplaza.

4.4.1. El motor Portescap.

Para la realización del proyecto se había comprado ya un motor lineal paso a paso, este es del fabricante Portescap, modelo 56DBM10B2U-L que es el que se ha tenido que emplear. Sus características más importantes son:

Característica	56DBM10B2U-L
Tensión (V)	12
Empuje max. (N)	124.6
Fuerza de sujeción (N)	88
Resolución de paso (mm/paso)	0.025
Pull-in rate max. (pasos/s)	275
Peso (kg)	0.454

Tabla 4.1: Características de Portescap 56DBM10B2U-L.

Si se multiplica la frecuencia máxima de pull-in por la resolución de paso se obtiene la velocidad lineal máxima del motor, que es 6.98 mm/s, muy inferior a la deseada por el INSIA. Y consultando la gráfica de empuje-frecuencia propia del modelo de motor en la figura 2.5 se ve que, la fuerza linear de este es un poco superior a la inercia de los “runners”, por lo que el motor los podrá desplazar pero no a la velocidad deseada. Además al tener un empuje tan ajustado a la inercia y ser un motor con baja frecuencia de paso, la elección de un perfil trapezoidal mejorara el desplazamiento de los “runners”, minimizando la posibilidad de perdida de pasos, pero empeoraremos mas si cabe la velocidad media del desplazamiento. Por ello para recuperar un mínimo de velocidad evitando mover más peso del necesario se dispone que el motor esté fijo y empujará únicamente la masa de los “runners”.

Un uso común si el tiempo de aceleración es significativo dentro del tiempo total de desplazamiento, para que el motor se desplace a la velocidad deseada es elevar la velocidad máxima del motor durante el periodo de velocidad constante, así se consigue ajustar la velocidad media del desplazamiento a la velocidad deseada, nunca alcanzando una velocidad que haga que debido a la efecto de masa de la carga el empuje del motor sea insuficiente y se pierdan pasos. En este caso esto no se puede realizar ya que se trabaja con la velocidad máxima del motor.

Como el motor del cual se dispone para la realización del proyecto no cumple con las especificaciones del mismo, los cálculos para la obtención del perfil de velocidad no son más que información ya que la velocidad deseada para el movimiento es mucho mayor a la del motor del que se dispone. Lo que se hace es fijar la distancia minima de 1 mm para obtener perfil de velocidad trapezoidal a velocidad constante. La variación de la frecuencia pasos a lo largo del tiempo durante el periodo de aceleración para el mejor manejo de la carga se hará por software en el desarrollo software.

Para realizar esta variación de tasa de pasos con el motor de Portescap, lo que se hace es alcanzar la frecuencia de pasos mayor realizándose este aumento de tasa pasos en tres tramos, el primer tramo de aceleración de 0 a 200 pasos/s disponiendo de un empuje de motor de 40 N. Un segundo tramo de 200 a 250 pasos/s con un empuje de 20 N. Y tramo final de 250 a 275 pasos/s con 12 N. Los tramos de frenado son los mismos, realizándose estas tasas de pasos en sentido inverso.

El cálculo del tiempo de aceleración se hace con las ecuaciones 2.6 y la gráfica de la figura 2.5. Aproximando la representación del pull-in rate a una recta T(f) y con la carga de los “runners”

siendo una constante igual a la inercia de estos $T_L(f)$. Se obtiene el tiempo que tardara el motor en alcanzar esa velocidad.

$$p = \frac{88-15.5}{275-0} = -0.263$$
$$T(f) = 89.1 - 0.263 f(N)$$
$$T_L(f) = 3.92(N)$$
$$t = 0.4 * 0.025\left[-3.80228\log(85.2 - 0.263 f)\right]_{f1}^{f2}$$
$$f_1 = 200 - 0(pasos/s) \rightarrow t_{acc1} = 36.5(ms)$$
$$f_2 = 250 - 200(pasos/s) \rightarrow t_{acc2} = 19.6(ms)$$
$$f_3 = 275 - 250(pasos/s) \rightarrow t_{acc3} = 15.7(ms)$$
$$t_{acc\_total} = t_{acc1} + t_{acc2} + t_{acc3} = 71.8(ms)$$

Ecuaciones 4.2: Cálculo de tiempo de aceleración de Portescap.

$T_L(f)$, en esta operación es únicamente la inercia de los "runners", es decir, el motor estará fijo desplazando los "runners" mediante el sinfín, el uso de esta disposición permite que la velocidad del movimiento sea mayor, al no tener que empujar también el peso del propio motor. A continuación se calcula el número de pasos a partir del cual se ha de cambiar la frecuencia de pasos.

$$N\_pasos\_acc = t_{acc} * f_{pasos}$$
$$N\_f_1 = 7.3 pasos$$
$$N\_f_2 = 4.9 pasos$$
$$N\_f_3 = 4.3 pasos$$
$$N\_pasos\_acc = 8 + 5 + 4 = 17 pasos$$

Ecuaciones 4.3: Cálculo del número de pasos por tramo Portescap.

Con estas condiciones la velocidad media que se dispone empleando un perfil trapezoidal con los tiempos de aceleración y frenado iguales se calcula empleando las ecuaciones 2.4, fijando un desplazamiento de 1mm, se obtiene un tiempo de velocidad constante y finalmente la velocidad media de la carga con las ecuaciones 2.6.

$$N\_total = \frac{r = 1(mm)}{mm/paso} = 40\,pasos$$

$$N\_v_{cte} = 40 - 2*17 = 6\,pasos$$

$$t\_v_{cte} = 6\frac{1}{f_3} = 21.81(ms)$$

$$t\_v_1 = 8\frac{1}{f_1} = 40(ms)$$

$$t\_v_{cte} = 5\frac{1}{f_2} = 20(ms)$$

$$t\_v_{cte} = 4\frac{1}{f_3} = 15.54(ms)$$

$$v_{media} = \frac{40*0.025}{2*71.8*10^{-3} + 21.8*10^{-3}} = 6.04(mm/s)$$

Ecuaciones 4.4: Cálculo de la velocidad media Portescap.

Ahora se sabe que con movimientos en tramos de 1 mm empleando este perfil de movimiento el motor desplazara los "runners" a una velocidad 6.04 mm/s, permaneciendo constante esta velocidad en desplazamientos más largos al realizarse este incrementando el número de tramos de 1 mm recorridos. En el apartado futuras líneas de trabajo se ha seleccionado un motor adecuado al sistema del cual se obtiene el perfil de velocidad para adecuarlo al sistema.

4.4.2. Driver DRV8805.

El DRV8805 ofrece una solución integrada para el control de motores paso a paso unipolares, con cuatro salidas con protección de sobre corriente proporcionada por diodos para capturar los transitorios generados por el apagado del motor. Además se puede colocar un diodo de retorno de corriente. Dispone de una lookup table que permite el control mediante una interfaz de paso/dirección. Tiene integrados tres modos de control para un motor unipolar. Posee también funciones internas de apagado para protección, indicándose este estado por un pin de salida del integrado.

Su tensión de operación puede variar entre 8.2 V y 60 V y proporcionando una corriente máxima de 1.5 A cuando conduce un único canal u 800 mA por canal cuando conducen los cuatro. La frecuencia de paso no debe superar los 250 kHz. Todas estas características hacen este driver apropiado para el control del motor Portescap.

Las señales en los pines del DRV8805 para el control del motor son:

- ***nENBL:*** Enable output, es la entrada de activación a nivel bajo de los drivers de salida.
- ***RESET:*** Es la entrada para realizar el reset de la lógica interna activo a nivel alto, lleva el índice al posición de partida.
- ***STEP:*** El motor se desplaza un paso con cada flanco de subida que se detecta en el pin.
- ***DIR:*** Con esta entrada se controla la dirección de desplazamiento del motor.
- ***SM0, SM1:*** Con estas entradas se selecciona el modo de control del motor.

SM1	SM0	MODO
0	0	Full-step.
0	1	Half-step.
1	0	Wave drive.
1	1	Reservado.

Tabla 4.2: Modos de control DRV8805.

- ***nFAULT:*** Salida activa a nivel bajo, para notificar un apagado de protección por sobre corriente o exceso de temperatura.
- ***nHOME:*** Salida que se pone a nivel bajo cuando el índice reinicia la cuenta de la secuencia de control. Las secuencias de control y nHOME se pueden ver en las tablas siguientes. Extraídas del datasheet del DRV8805. Es una señal igual a la generada por un ENCODER.

Es un encapsulado Wide SOIC de 20 pines, a continuación se puede ver el diagrama de bloques del dirver.

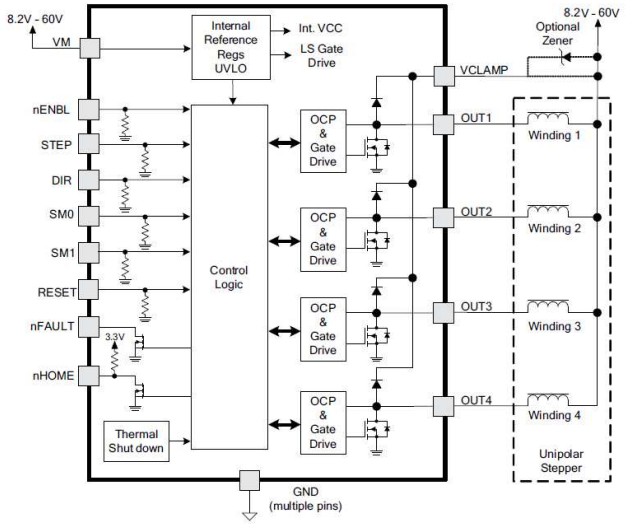


Figura 4.8: Diagrama DRV8805.

Las secuencias de excitación para el control del motor se tienen en las siguientes tablas, junto con la configuración de las señales de control.

Function	Step	RESET	DIR	STEP	nHOME	OUT1	OUT2	OUT3	OUT4
Reset	1	1	X	X	0	ON	OFF	OFF	ON
CW	2	0	1	↑	1	ON	ON	OFF	OFF
CW	3	0	1	↑	1	OFF	ON	ON	OFF
CW	4	0	1	↑	1	OFF	OFF	ON	ON
CW to home	1	0	1	↑	0	ON	OFF	OFF	ON
CCW	4	0	0	↑	1	OFF	OFF	ON	ON
CCW	3	0	0	↑	1	OFF	ON	ON	OFF
CCW	2	0	0	↑	1	ON	ON	OFF	OFF
CCW to home	1	0	0	↑	0	ON	OFF	OFF	ON
Hold	X	0	X	↑̄	no chg	no chg	no chg	no chg	no chg

Tabla 4.3: Secuencia paso completo (full step).

Function	Step	RESET	DIR	STEP	nHOME	OUT1	OUT2	OUT3	OUT4
Reset	1	1	X	X	0	ON	OFF	OFF	OFF
CW	2	0	1	↑	1	ON	ON	OFF	OFF
CW	3	0	1	↑	1	OFF	ON	OFF	OFF
CW	4	0	1	↑	1	OFF	ON	ON	OFF
CW	5	0	1	↑	1	OFF	OFF	ON	OFF
CW	6	0	1	↑	1	OFF	OFF	ON	ON
CW	7	0	1	↑	1	OFF	OFF	OFF	ON
CW	8	0	1	↑	1	ON	OFF	OFF	ON
CW to home	1	0	1	↑	0	ON	OFF	OFF	OFF
CCW	8	0	0	↑	1	ON	OFF	OFF	ON
CCW	7	0	0	↑	1	OFF	OFF	OFF	ON
CCW	6	0	0	↑	1	OFF	OFF	ON	ON
CCW	5	0	0	↑	1	OFF	OFF	ON	OFF
CCW	4	0	0	↑	1	OFF	ON	ON	OFF
CCW	3	0	0	↑	1	OFF	ON	OFF	OFF
CCW	2	0	0	↑	1	ON	ON	OFF	OFF
CCW to home	1	0	0	↑	0	ON	OFF	OFF	OFF
Hold	X	0	X	↑̄	no chg	no chg	no chg	no chg	no chg

Tabla 4.4: Secuencia medio paso (half step).

Function	Step	RESET	DIR	STEP	nHOME	OUT1	OUT2	OUT3	OUT4
Reset	1	1	X	X	0	ON	OFF	OFF	OFF
CW	2	0	1	↑	1	OFF	ON	OFF	OFF
CW	3	0	1	↑	1	OFF	OFF	ON	OFF
CW	4	0	1	↑	1	OFF	OFF	OFF	ON
CW to home	1	0	1	↑	0	ON	OFF	OFF	OFF
CCW	4	0	0	↑	1	OFF	OFF	OFF	ON
CCW	3	0	0	↑	1	OFF	OFF	ON	OFF
CCW	2	0	0	↑	1	OFF	ON	OFF	OFF
CCW to home	1	0	0	↑	0	ON	OFF	OFF	OFF
Hold	X	0	X	↑̄	no chg	no chg	no chg	no chg	no chg

Tabla 4.5: Secuencia wave drive.

La elección de este driver es debida a su facilidad de manejo y conexión, es capaz de excitar adecuadamente las bobinas del motor y a que la señal nHOME se emplea para simular la señal que debería generar un ENCODER situado en el eje del motor para realimentar su posición contando los desplazamientos que se realizan teniendo un control en bucle cerrado. Ya que no se dispone de un ENCODER en el eje del motor y tampoco se dispone de una varilla sin fin apta

para el uso del motor en la cual situar el dispositivo adecuado. Esta señal nHOME es ideal para simular el ENCODER y mantener el control de la posición del paso a paso.

4.4.3. Esquemático de la conexión.

El esquemático de la conexión es sencillo, directo al micro hay que poner una resistencia de pull-up en el pin nFAULT, en serie hay un diodo para que sea visible este apagón del driver por protección de sobre corriente o temperatura.

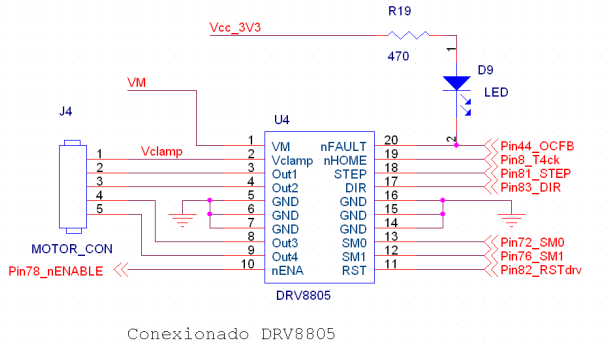


Figura 4.9: Esquemático de conexión del DRV8805.

4.5. Conexión al Bus CAN.

4.5.1. Transciver MCP2551.

El MCP2551 es un transciver CAN de alta velocidad que sirve de interfaz entre el controlador CAN y el bus físico; proporciona la posibilidad de recepción y transmisión diferencial, es compatible con la norma ISO-11898, pudiendo operar a velocidades de hasta 1 Mb/s. También proporciona un buffer entre el controlador CAN y los picos de tensión que se pueden generar en el bus por fuentes externas hasta +/-250 V y una protección por cortocircuito de los cables del bus hasta +/-40 V que deshabilitan los drivers de salida, permaneciendo el resto de bloques operativos. Ofrece una protección de reset y una brown-out, así como la detección de bit dominante permanente para asegurar que un nodo defectuoso no perturbe el estado del bus.

El transciver MCP2551 realiza tres operaciones:

- ***Transmitir:*** El controlador emite una serie de datos lógicos hacia la entrada TXD del transciver, el correspondiente estado dominante o recesivo será puesto en el bus mediante los pines CANH y CANL.

- ***Recibir:*** El transciver recibe bits o estados dominantes o recesivos en los pines CANL CANH. Estos estados son generados en la salida del pin RXD hacia el controlador CAN para que este reciba la trama.
- ***Traducción de estados:*** Un '1' lógico en la entrada TXD produce un estado recesivo en el bus. Un '0' lógico en la entrada TXD produce un estado activo.

El encapsulado es DIP o SOIC 8, la descripción de los pines para la comunicación CAN es:

- ***TXD:*** El controlador pone datos para transmitirlos al bus CAN.
- ***RXD:*** Salida hacia el controlador CAN de la trama recibida por el bus.
- ***Vref:*** Es una tensión de referencia Vdd/2.
- ***CANL:*** Conexión al cable CAN_L del bus.
- ***CANH:*** Conexión al cable CAN_H del bus.
- ***Rs:*** Se ha de conectar la resistencia de selección de pendiente.

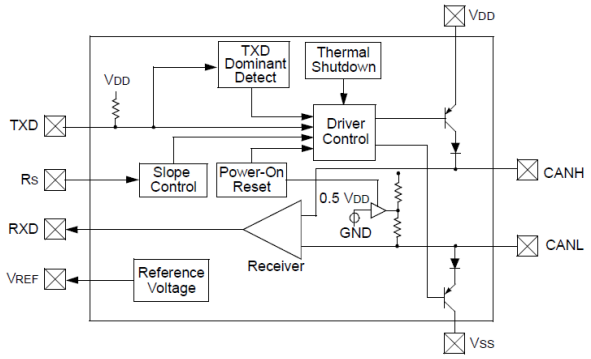


Figura 4.10: Diagrama de bloques del MCP2551.

4.5.2. Esquemático de la conexión.

Rs, en el circuito R8 controla la rampa de la señal generada por el MCP2551 en el bus.

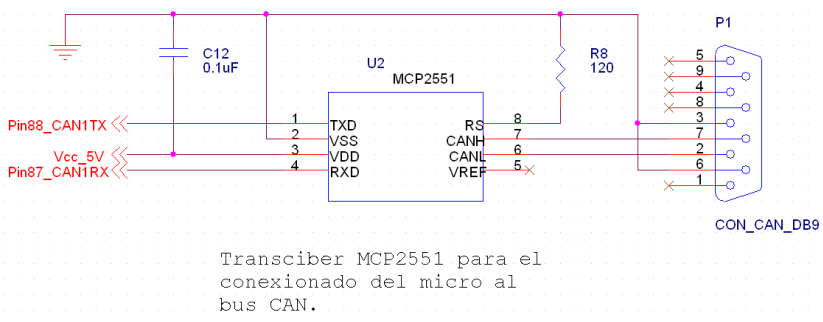


Figura 4.11: Conexión del MCP2551.

El Puerto E se emplea para notificar mediante leds el estado del nodo, activo, pasivo o bus OFF.

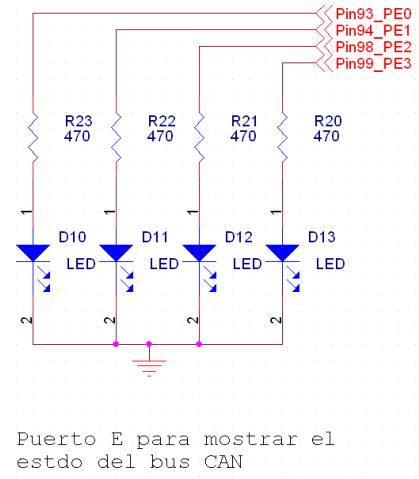


Figura 4.12: Esquemático del puerto E.

4.6. Sensor de Posición del Acelerador

Existe un TPS (Throttle Position Sensor) en el monoplaza, es un potenciómetro con la curva característica lineal vista en la figura 1.8 proporcional al ángulo de giro de la válvula del acelerador con una sensibilidad (S) de 50mV/%. Según lo que se especifica en el punto 1.8 en referencia al uso del TPS, bastaría con comparar el nivel de tensión puesto por el TPS con una tensión 3.25 V equivalente al 65% de la máxima generada por este sensor, para detectar si se deben incrementar los “runners” o no.

Para dar un mejor rendimiento, lo que se va ha hacer es aprovechar el ADC de 10 bits del microcontrolador para saber el nivel de tensión del sensor acelerador. Este tiene una resolución de 3.22 mV (1LSB), capacidad sobrada para medir un incremento de un 1%.Y así disponer de esta información para el control software y se pueda disponer de ella por parte de cualquier otro módulo del coche al introducirla al bus CAN en la trama de datos.

4.6.1. Diseño circuito acondicionador.

Para poder realizar la lectura con el ADC se ha de bajar el nivel de tensión máxima del TPS al nivel de tensión máximo de AVdd, si no se realizara la conversión la conversión es imposible, para ello se a optado por el diseño de un amplificador diferencial de ganancia fija, sin ajuste de offset y con alta impedancia de entrada.

La selección de este circuito como acondicionador así como su configuración se basa en que no prima saber el valor exacto del TPS, se evita el uso de componentes que debido a vibraciones puedan desajustarse; y, como el TPS es un potenciómetro el circuito acondicionador ha de ser transparente a este potenciómetro. Como lo que se desea es acondicionar la señal de 5 V del TPS al nivel de tensión de alimentación AVdd del PIC32 que es de 3.3 V. Se ha de tener una ganancia (G) de 0,66 V/V. Como se advierte viendo la figura 4.13 al hacerse R13 = R15 = R1 y R14 = R16 = R2, la ganancia del circuito acondicionador se obtiene con las siguientes ecuaciones:

$$S = \frac{\Delta Vo}{\Delta \%}$$

$$G = \frac{\Delta Vo}{\Delta Vi}$$

$$G_{circuito} = \frac{R_2}{R_1}(V_{TPH+} - V_{TPH-})$$

$$EG = G_{Real} - G_{ideal}$$

$$Err(mV) = Vos - Voideal$$

$$Err(\%) = \frac{Err(V)}{S}$$

Ecuaciones 4.5: Cálculo del análisis del circuito acondicionador.

Para el amplificador diferencial se ha empleado el amplificador operacional OP-07 por sus buenas características de entrada. Y para los seguidores de tensión que aíslan el circuito acondicionador del TPS se ha empleado el LM358 por la integración de dos amplificadores operacionales en un único integrado, ambos componentes son DIP8.

Finalmente, ya que internamente el PIC32 permite seleccionar una tensión de referencia interna o externa para la conversión analógico-digital, se ha implementado un circuito para poder ajustar esta tensión de referencia del conversor ADC y así si fuera necesario tener un punto de ajuste de la ganancia del circuito acondicionador se seleccionaría esta tensión de referencia externa para la conversión analógico-digital.

4.6.2. Características del circuito acondicionador.

Se analiza la validez del circuito acondicionador a partir de las ecuaciones anteriores, se busca una ganancia ideal de 0.66 V/V para ello se van a realizar medidas cada 10 % del TPS para comprobar la sensibilidad del circuito. Con los datos de las medidas se forma la tabla siguiente:

TPS real (V)	Acelerador (%)	TPS ideal (V)	Vacond ideal (V)	Vacond Real(V)	Err (mV)	Err (%)
0,00	0	0	0	0,00	0,70	0,02
0,50	10	0,5	0,33	0,34	6,00	0,18
1,00	20	1	0,66	0,67	12,00	0,36
1,50	30	1,5	0,99	1,01	18,00	0,53
2,00	40	2	1,32	1,34	22,00	0,65
2,50	50	2,5	1,65	1,68	27,00	0,80
3,00	60	3	1,98	2,02	35,00	1,04
3,50	70	3,5	2,31	2,35	39,00	1,15
4,00	80	4	2,64	2,69	49,00	1,45
4,50	90	4,5	2,97	3,03	63,00	1,86
5,00	100	5	3,3	3,38	80,00	2,37

Tabla 4.6: Evaluación del circuito acondicionador TPS.

Se comprueba que existe un error de ganancia de 0.01 (V/V). Y un error en la medida máximo de 2,37% en la posición del acelerador. Con un ajuste de ganancia este error se reduce y el empleo la tensión de referencia externa permitirá este ajuste. Otra posibilidad para resolver esta incidencia es el ajuste software, utilizando el error de ganancia para realizar el cálculo de ajuste.

Y una consideración importante para la conexión del TPS es la tensión en modo con la que el circuito puede operar con normalidad, siendo esta:

$$-12+0.66\frac{|V_{TPS}|}{2}<Vc<12-0.66\cdot\frac{|V_{TPS}|}{2}$$

Ecuaciones 4.6: Tensión en modo común del circuito acondicionador.

4.6.3. Esquemático de la conexión.

También se han puesto en las entradas del micro unos diodos y unas resistencias para la protección de los pines analógicos, como se observa en el siguiente esquemático.

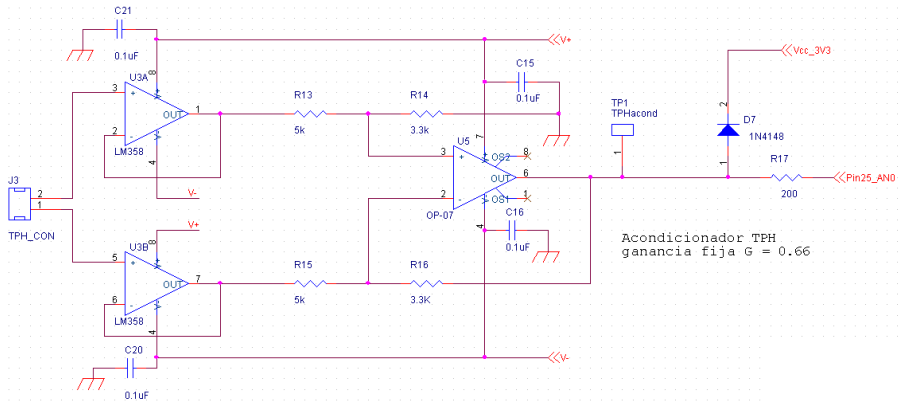


Figura 4.13: Esquemático amplificador diferencial.

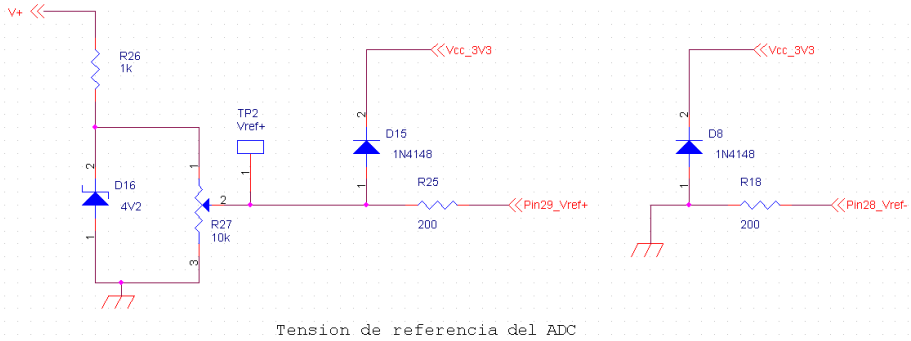


Figura 4.14: Esquemático tensión de referencia.

4.7. Sensor del Árbol de Levas.

En el caso del Sensor del Árbol de Levas para la captación de las rpm se emplea un sensor de Efecto Hall. Para su uso es necesaria una pieza dentada o un engranaje de un material ferromagnético según un determinado patrón que se calcula en función del perímetro de dicha pieza.

La pieza se fija al árbol de levas, de forma que gire solidariamente al mismo y se introduce el sensor a la distancia adecuada de esta rueda dentada para medir la variación del campo magnético que se produce al girar, esta distancia nos la da el parámetro Air Gap del sensor. En el caso del ATS616, esta distancia va de 0.4 < Air Gap < 2.5 mm. El giro hace que se genere la señal digital en la salida, que cambia de nivel cada vez que detecta un pico en el campo magnético, el pico detectado es la característica Bqp del componente.

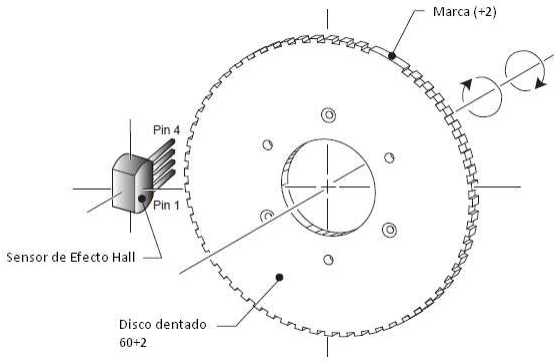


Figura 4.15: Sensor de Efecto Hall y engranaje.

Una relación muy cómoda seria hacer que tenga 60 dientes para que cuando el motor de una vuelta completa se hayan producido 60 pulsos. En consecuencia si el motor hace una revolución en un minuto, el árbol de levas, que gira solidario al motor también lo hará junto con el disco dentado. Con el sensor situado a la distancia adecuada, este generara entonces una señal de 1 Hz

de frecuencia. Por lo tanto la sensibilidad de este sensor esta fijada en mayor medida por el conjunto sensor + engranaje.

4.7.1. Sensor de efecto Hall ATS616.

El ATS616 es un sensor detector de pico por diente de engranaje con control automático de ganancia y un condensador integrado para proporcionar detección de flancos de engranajes precisos. Su encapsulado es de una carcasa plástica de alta temperatura, que mantiene a salvo sus componentes magnéticos. La tecnología utilizada para este circuito se basa en el efecto Hall. Su filtro detector de pico elimina el offset. Este componente esta pensado para aplicaciones que requieran una medida precisa de ciclo de trabajo o detección de flanco, como es detectar velocidad de giro del árbol de levas.

Debido a la frecuencia máxima de funcionamiento de este dispositivo (10kHz) y para poder medir el rango de rpm deseado, este engranaje podría tener 30 dientes, resultando una sensibilidad de 0.5 Hz/rpm. Supuesto que se sigue en adelante y se simula con el generador de señal.

En un encapsulado SIP4, su pin-out es el siguiente:

- ***Vcc:*** tensión de alimentación de 3.5 V a 24 V.
- ***Vout:*** Señal digital variable en función del campo magnético detectado entre el sensor y el engranaje. Sus niveles de tensión son 0 V y Vcc.
- ***Test:*** Que debe unirse a masa.
- ***Gnd:*** Conexión de la masa.

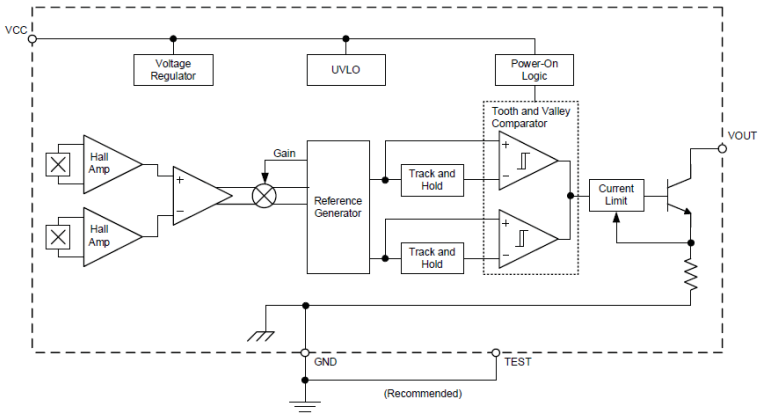


Figura 4.16: Diagrama de bloques del ATS616.

4.7.2. Esquemático de la conexión.

Para la conexión del sensor de efecto Hall al microcontrolador solo es necesario conectar una resistencia de pull-up a la patilla Vout del integrado. En el diseño hardware, se ha dejado un jumper para conectar esta resistencia ante la posibilidad del uso de otro sensor que no necesite de esta resistencia.

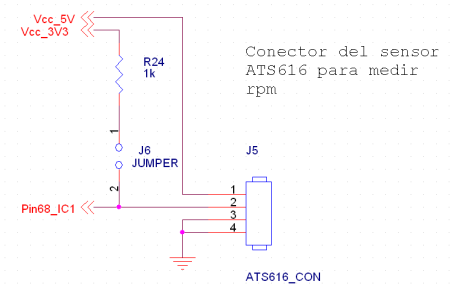


Figura 4.17: Esquemático de la conexión del ATS616.

4.8. Sensor Fin de Carrera.

SFC es un acrónimo de Sensor de Final de Carrera. Este sensor es un pulsador que se conecta a la interrupción externa 3 del PIC32 y se sitúa en los extremos de los recorridos del "runners". Esto da robustez al sistema evitando que el motor paso a paso intente avanzar hacia donde ya no puede. Y además se tiene una referencia para el posicionado inicial del motor tras un encendido o en caso de perdida de su posición.

4.8.1. Esquemático de la conexión.

En el esquemático solo se está el conector al cual se conectarían dos pulsadores en paralelo, uno para el extremo superior del recorrido y otro para el inferior mediante cables, para poder situar estos sensores a la distancia que se desee del circuito de control.

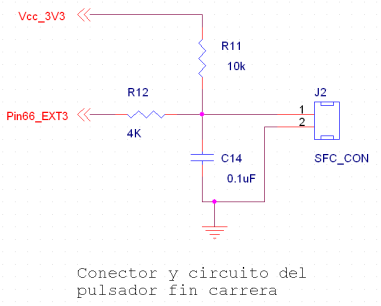


Figura 4.18: Esquemático del SFC.

4.9. Botón ON/OFF.

Se trata de un pulsador para que en caso de mal funcionamiento, o por cualquier otro motivo se desease desactivar la acción del VAI haya una forma sencilla y accesible de hacerlo, ya que es posible que por situaciones que se puedan dar durante la realización de las pruebas en la competición resulte mejor desconectar este sistema.

4.9.1. Esquemático de la conexión.

El esquemático es el mismo que para el SFC, con la diferencia de que en este el pulsador si aparece.

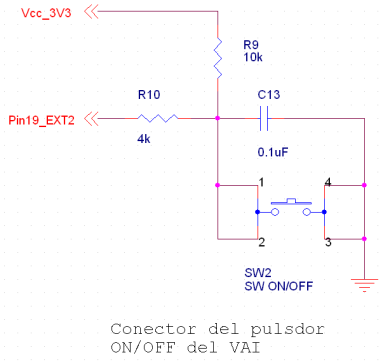


Figura 4.19: Esquemático del pulsador ON/OFF.

5. Desarrollo software.

5.1. Diagrama de bloques software.

El software del módulo esta dividido en cuatro bloques, uno de ellos es la adquisición de datos, (RPM y TPS), otro la comunicación con el bus CAN. El tercero es el control del motor, que se apoya en un conjunto de periféricos para el control del driver y el motor paso a paso, que necesita de información exterior al microcontrolador. Y por ultimo un control VAI, es el que recibe la información de los sensores, del bus CAN, de la memoria y actúa sobre el control del motor, para responder con una posición adecuada de los "runners" respecto a la información de la que dispone y envía información al bus.

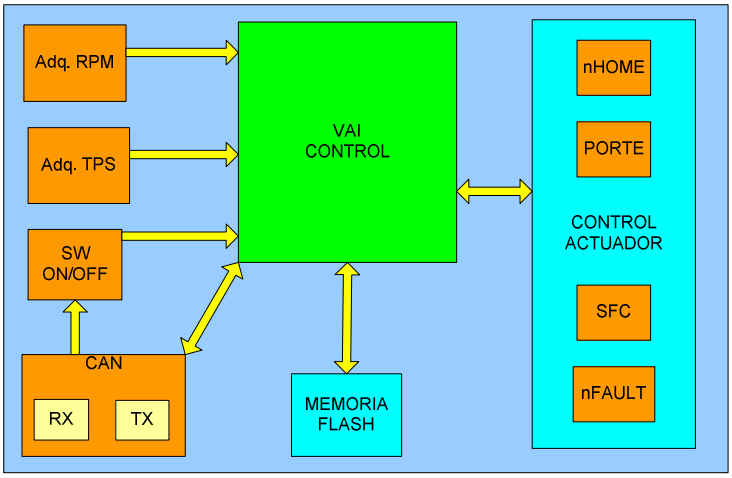


Figura 5.1: Esquema de trabajo software.

A cada una de estas partes le corresponden uno o más periféricos.

- ***Control VAI:*** Control Variable Air Intake, es un programa que se encarga del gobierno del sistema controlando como se actúa sobre el motor, gestiona las tablas de posicionamiento y los mensajes enviados y recibidos por el bus CAN. Además del acceso a memoria no volátil. Usa el timer 5.
- ***Adquisición rpm:*** Programación del IC (Input Capture) para la lectura del sensor de rpm del motor de combustión.
- ***Adquisición TPS:*** Programación del ADC para la conversión de la señal del TPS.
- ***CAN Tx/Rx:*** Configuración de la comunicación CAN y de las FIFO, gestión de errores y estado del bus.

- ***SW ON/OFF:*** Es un interruptor software, que pude ser "pulsado" por CAN o por la interrupción externa 2 (EXT2).
- ***Control motor:*** Se disponen de varios periféricos del microcontrolador para el control del movimiento del motor paso a paso, el puerto D, la interrupción externa 3 (EXT3) que actúa como SFC (Sensor de Fin de Carrera), Output Compare 5(OC5) para generar la frecuencia de paso y captar el estado nFAULT, además este periférico utiliza el timer 2 para generar esta frecuencia y el perfil de velocidad adecuado. Y el timer 4 para la adquisición de la señal de realimentación proveniente de un ENCODER, simulado con la señal nHOME.

Hay que saber como son las señales que estos periféricos van a recibir para realizar su configuración, por ello hay que atender al hardware que se dispone.

5.2. Entorno MPLAB.

La versión empleada para el desarrollo del proyecto es MPLABX IDE. Es un programa de software que se emplea para el desarrollo de aplicaciones para microcontroladores PIC. Es un entorno de desarrollo integrado (IDE), que proporciona un único "medio ambiente" para el desarrollo de códigos para microcontroladores, puesto que este permite la depuración, compilación y los medios necesarios para generar ejecutables y programarlos en microcontroladores PIC. Para poder hacer la programación y depuración en el microcontrolador se dispone del PICkit3, que conecta el PC al circuito por medio de un programador/depurador USB.

Como el microcontrolador seleccionado es de 32-bit, el compilador que se ha de instalar en el IDE debe ser el dedicado a microcontroladores de esta arquitectura.

5.3. Interrupciones PIC32.

Como es sabido, en los sistemas empotrados las distintas interrupciones tienen unos vectores, para realizar el salto hacia la rutina de interrupción, en la mayoría de los casos el orden que tengan estas interrupciones en dichos vectores es la prioridad que tienen, siendo más prioritarios cuanto más cerca del inicio de la memoria están.

Existen diversos métodos para alterar esta prioridad según cada fabricante. En este caso, los microcontroladores PIC32 pueden funcionar de dos formas. Con un único vector de interrupción en el cual la aplicación software tendrá que identificar la fuente de interrupción y actuar en consecuencia, es decir, cualquier interrupción habilitada producirá un salto hacia una única

rutina de interrupción. Esto pude ser útil en aplicaciones simples. La otra forma de funcionamiento es con múltiples vectores, donde cada vector de interrupción realiza un salto hacia su rutina de interrupción. Aún en modo multivector existen vectores de interrupciones distintos que van a una misma dirección de interrupción, por ejemplo para el control de una UART existen 3 vectores de interrupción: recepción, transmisión y error; pero las tres fuentes de interrupción saltan a la misma dirección de interrupción.

Seleccionando este ultimo modo de funcionamiento, la prioridad por defecto en las interrupciones viene dada por su orden, pero PIC32 dispone de grupos de prioridades y subprioridades. Así a una interrupción se le debe asignar un grupo de prioridad, y hay 7 grupos en total, siendo el grupo más bajo el 1 y el más alto el 7. Y una subprioridad separada en cuatro niveles, siendo el 0 el más bajo y 3 el más alto. El grupo de prioridad sirve, para que cuando el microcontrolador está atendiendo una interrupción y se produce otra perteneciente a un grupo más prioritario el contador de programa del microcontrolador saltará a la rutina de esta nueva interrupción; los grupos de prioridad más alta son atendidos en primer lugar interrumpiendo al los de menor grupo de prioridad. La subprioridad se emplea en el caso que estén pendientes por atender dos interrupciones del mismo grupo, se atenderá en primer lugar la interrupción con la subprioridad mayor, atendiéndose la otra después. Si dos interrupciones tienen igual prioridad y subprioridad, se atenderá primero a la interrupción cuyo vector de interrupción sea más bajo.

Con esta característica del microcontrolador, se controla el flujo del programa. Por ejemplo, en el desarrollo de este proyecto este hecho es de ayuda ya que para medir la frecuencia del sensor RPM la interrupción del Input Capture debe atenderse lo más rápido posible para que el error en la medida de frecuencia sea menor. Al introducir esta interrupción en un grupo más alto que el resto se garantiza que siempre se entenderá esta interrupción primero independientemente de que estuviera haciendo el microcontrolador. Las interrupciones en el programa VAI tienen el siguiente orden de prioridades, con la más alta en la primera posición de la tabla:

interrupción	Vector	Grupo de prioridad	Subprioridad
IC1	5	6	0
OC5	22	4	0
TMR4	16	3	1
TMR2	8	3	0
TMR5	20	3	0
EXT1	7	2	0
EXT2	11	2	0
ADC	33	1	0
CAN	58	1	0

Tabla 5.1: Prioridades del módulo VAI.

5.4. Oscilador.

La configuración del oscilador es necesaria para fijar la velocidad de operación. PIC32 permite una variada configuración de oscilador:

- Tiene 4 fuentes para el oscilador, interna y externas.
- Posee 2 PLL integrados dedicados a tareas distintas (USB, oscilador).
- Se pueden multiplicar, o dividir la frecuencia en la entrada del PLL y dividir su salida, para seleccionar una frecuencia de operación a partir de una fuente interna o externa
- Un divisor de frecuencia, para la fuente seleccionada.
- Mediante el control software se puede cambiar entre las distintas fuentes de reloj.
- También tiene un monitor del reloj, para permitir a la aplicación un apagado seguro si detecta algún fallo.

El diagrama de bloques del oscilador de PIC32MX, es el siguiente:

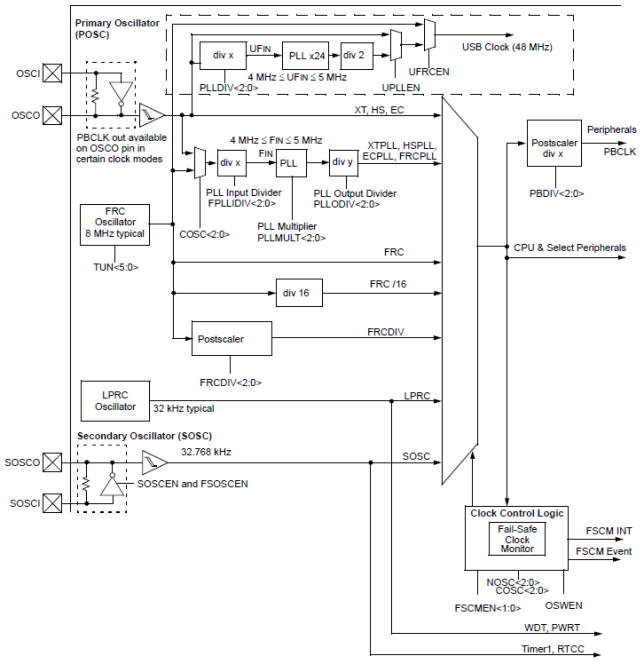


Figura 5.2: Diagrama del oscilador.

Como se puede ver existen dos frecuencias generadas por el oscilador. La frecuencia de la CPU y algunos periféricos elegidos y la frecuencia de periféricos, propia de la mayoría de los periféricos. Para la configuración de estas dos frecuencias se ha de acceder a los registros SFR dedicados al oscilador. La configuración de estos registros para el sistema VAI se realiza

buscando una elevada frecuencia de trabajo para una buena medida de rpm y alcanzar frecuencias altas con el PWM (OC5). En el ANEXO II se ve el código de esta configuración de frecuencias, que es la siguiente:

- Frecuencia CPU = 80 MHz.
- Frecuencia periféricos =10 MHz (PBCLK).

5.5. Pines de entrada salida.

Los pines de entrada salida pueden considerarse los periféricos más simples. Ellos permiten al microcontrolador PI32MX monitorizar y controlar otros dispositivos. Para añadir flexibilidad y funcionalidad al integrado, varios pines tienen multiplexadas diferentes funciones, estas funciones dependen de qué periféricos dispone el dispositivo y cuales están asociados a un determinado pin.

Las funciones comunes a la mayoría de estos pines son:

- Configuración open-drain para salida.
- Configuración pull-up para entrada.
- Una entrada monitorizada puede generar una interrupción.
- Rápida manipulación de sus bits en los registros.

Los pines de entrada salida deben ser configurados adecuadamente a la interrupción a la que están asociados, aunque algunas interrupciones en su activación configuran sus propios pines, esta práctica resulta aconsejable para un mejor control de los pines disponibles.

La configuración de estos pines se realiza en la función setup_micro( ) que se ve en el ANEXO II.

5.6. Variable Air Intake (VAI).

El control de todo el módulo VAI es una maquina de estados, y las transiciones entre estados se pueden producir por el resultado de la operación del estado actual o por la acción de alguna interrupción. Esta maquina tiene 10 estados diferentes, para gestionar el control motor, la comunicación CAN y los accesos a memora FLASH. Como se ve en la figura de la máquina de estados, la zona azul representa a los estados en los que el motor paso a paso se esta moviendo o

esperando para moverse y la amarilla la parte de comunicación CAN, cuando después de alguna recepción CAN se devuelve el motor paso a paso al estado anterior se representa llevando la flecha de transición a este circulo azul.

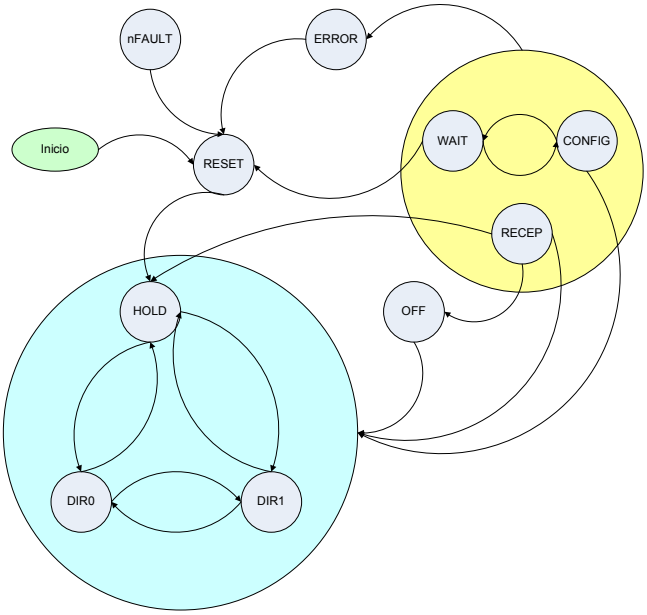


Figura 5.3: Maquina de estados de la función main.

En cada ejecución del bucle de la maquina se evalúa si hay información disponible para calcular las rpm del motor de combustión, además, el timer 5 se emplea para tener un periodo de envío por bus CAN menor a un incremento de 100 rpm del monoplaza, según la información facilitada por el INSIA este tiempo debe ser 14.5 ms o menor según se ve en el apartado 1.8. Con esto se quiere decir que se mide y se envía constantemente la información de los sensores al bus CAN, exceptuando cuando se esta configurando alguna tabla.

Para el control de las posiciones del motor paso a paso se emplea una tabla con la información de las posiciones de los "runners" controlados por el motor paso a paso. Según la configuración de dicha tabla se conseguirá una respuesta en potencia diferente del monoplaza, es decir, con cada tabla se varía la geometría del motor. En automoción, a estos cambios de geometría se los conoce como mapas de motor, por lo que cada tabla produce un mapa de motor distinto. Se ha dispuesto que se puedan almacenar en memoria no volátil hasta 6 mapas de motor diferentes, uno por cada marcha que dispone el vehiculo. La tabla es un array de estructuras formadas por 3 tipos enteros.

- ***RPM:*** Es un entero de 16 bits, es el umbral máximo de rpm para una posición de una tabla de mapa motor.

- ***mm:*** Es la distancia en mm que hay desde la posición de reposo de los "runners" y la posición donde se deben permanecer los mismos, mientras no cambien las condiciones de rpm y TPS leídas. Es un entero de 8 bits.
- ***MAXPOSC:*** Un entero de 8 bits que indica el número máximo de posiciones de la tabla.

A continuación se explican los distintos estados del autómata, la codificación se encuentra en el ANEXO II. Algunos elementos para la comunicación CAN se describen en la sección propia.

5.6.1. nFAULT.

En este estado la máquina de estados ha detectado la situación de nFAULT con la interrupción del Output Compare 5 del driver del motor paso a paso, en este estado se sondea el DRV8805 para saber cuándo se sale de esta situación y cambiar de estado hacia el RESET.

5.6.2. RESET.

Con este estado se consigue un reset ordenado con la ayuda de la variable RSTStatus para, además, ir notificando por CAN la evolución del reset del módulo, reiniciando por separado, algunos periféricos, la tabla de mapa motor y TPS mínimo, y la posición de los "runners". O una combinación de estos. La salida de este estado siempre es hacia el estado HOLD.

5.6.3. HOLD.

En este estado se controla cuando el motor paso a paso debe arrancar y llevar los "runners" a la posición adecuada. Si las rpm leídas son mayores que las de la tabla de la posición actual y se está actuando sobre el TPS más de un mínimo establecido y aún queda alguna posición por subir, se mueven los "runners" hacia la siguiente posición y se cambia el estado a DIR1. Si por el contrario las rpm leídas son menores que el umbral de la posición anterior y aún pude bajar una posición de la tabla, se mueven los "runners" a la posición anterior y el cambio de estado es a DIR0.

5.6.4. DIR1 y DIR0.

Estos estados permiten saber la dirección de desplazamiento de los "runners" y facilitan el funcionamiento del bloque Control Motor. En estos estados el bloque Control Motor desplaza los "runners", pero en la función principal no se hace nada.

5.6.5. WAIT.

Estado al que se accede después de la recepción de alguna orden procesada con el SID1, en este estado se gestionan las respuestas que se transmitirán al bus CAN desde el estado CONFIG, en función de la acción hecha durante esta configuración. Y se controla la transición hacia RESET cuando se haya terminado o se hayan restaurado los valores anteriores en caso de error.

5.6.6. RECEP.

Este estado gestiona la recepción de las órdenes recibidas con SID2, si se recibe una orden que no está configurada, se devuelve el autómata al estado anterior de desplazamiento. Las órdenes posibles son:

- ***RSTCAN:*** Es la orden de reset del módulo por bus CAN.
- ***VAISW:*** Cuando se recibe esta orden se acciona sobre el "interruptor" del VAI, llevando el autómata al estado OFF o a recuperar su movimiento anterior dependiendo del estado en el que estuviera dicho interruptor.
- ***CHMAPA:*** Esta orden realiza el cambio a un mapa motor distinto, requiere del uso del campo "extra" del tipo de DATORX en el cual se debe indicar el número de mapa de motor a cargar, del 0 al 5. Si se envía el número de marcha junto con esta orden se consigue que se use un mapa motor según la marcha que se este utilizando en el monoplaza.

5.6.7. CONFIG.

Este estado también es de recepción CAN, pero en este caso las órdenes recibidas tienen SID1. Se paran las transmisiones constantes y solo se transmite una respuesta en el campo info de la variable CANStatus, para notificar la evolución de la creación o modificación de una tabla. Si se recibe una orden que no está configurada, se devuelve el autómata al estado anterior.

- ***NTABLA:*** Es la orden por bus CAN que se ha de enviar con cada posición de tabla enviada durante la creación de una tabla. La respuesta en el campo info es un 3 cuando indica orden realizada, un 2 notifica un error en la posición enviada al crear la tabla, un 1 indica tabla en creación y que la posición enviada es correcta y 0 es error al crear la tabla y recuperación de la tabla anterior.

- ***MTABLA:*** Con esta orden se modifica una posición de la tabla que este en uso, la respuesta en el campo info en este caso es un 3 si la posición se modifico correctamente y es 0 si no se pudo realizar la orden.
- ***MTPS:*** Esta orden modifica el valor del TPS mínimo (TPSMIN) que se debe estar actuando para que los "runners" decrementen su longitud. Un 3 en el campo info del status indica que el nuevo TPS ha actualizado y 0 indica que no se modificó y se recuperó el anterior.

5.6.8. ERROR.

Es un estado en el que si se produjo overflow en alguna de las FIFO de recepción se notifica por CAN, en caso de un error en el módulo CAN se llena el campo status de CANBufferTx con '1' para comunicar este hecho. Esta variable es un tipo de estructura definida (DATOTX), empleada para la transmisión de datos en el bus CAN.

5.6.9. OFF.

A este estado se accede cuando se pulsa el interruptor VAI o se envía la orden VAISW con el SID2, en este estado no se actúa sobre el motor paso a paso, por lo que los "runners" no se mueven, pero se sigue transmitiendo normalmente por el bus CAN los datos adquiridos.

5.7. CAN.

El módulo se comunica por bus CAN con el resto de dispositivos conectados a él, empleando este protocolo de comunicación serie asíncrona. La familia PIC32MX integra hasta dos módulos CAN. Estos módulos tienen las siguientes características:

- Configuración.
  - Especificación Full CAN 2.0B.
  - Tasa de bits programable hasta 1Mbps.
- Recepción y transmisión.
  - 32 FIFOs para mensajes.
  - Cada FIFO puede almacenar hasta 32 mensajes pudiéndose guardar un total de 1024 mensajes.
  - Las FIFO pueden ser usadas para transmisión o de recepción.
  - 32 filtros para aceptar mensajes.

- 4 registros de mascaras de aceptación para el filtrado.
- Respuesta automática a Trama Remota (TRT).
- Soporte para direccionamiento DeviceNet.

- Adicionales.
  - Modos loopback y solo escucha para test, diagnostico y monitorización del bus.
  - Operaciones en baja potencia.
  - El módulo CAN es bus Master del sistema de buses de PIC32.
  - No requiere el uso de canales DMA para su funcionamiento (ya dedicados).
  - Modo de recepción solo de datos.
  - Timer dedicado para el time stamp.

Los buffers de mensajes residen en la RAM del dispositivo, no hay buffers dentro del módulo CAN. El módulo CAN no necesita el uso de Acceso Directo a Memoria (DMA) para manejar mensajes en la RAM, sin la intervención de la CPU ya que tiene un canal DMA ya asignado. En la figura se ve el diagrama de bloques del CAN en PIC32.

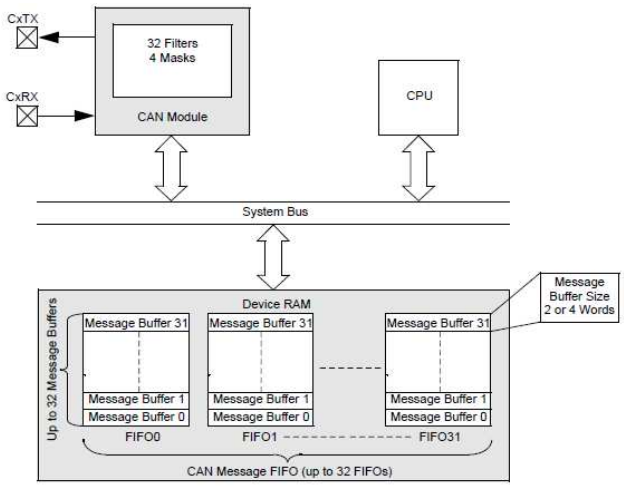


Figura 5.4: Diagrama de bloques del módulo CAN.

Las interrupciones del módulo CAN se clasifican en tres capas, las interrupciones de las capas bajas se propagan hacia las capas superiores, multiplexándose en una única interrupción por cada módulo CAN. Entonces una interrupción de la capa 1 es la multiplexación de varias interrupciones de la capa 2, y algunas interrupciones de esta segunda capa son la multiplexación de interrupciones de la capa 3.

Siguiendo la figura 5.5 la primera capa genera una interrupción en la CPU. Esta es causada por alguna de la segunda y esta a su vez puede ser causada por alguna de la tercera capa. El módulo

CAN puede habilitar o no cada una de las interrupciones de las tres capas mediante el control de interrupciones (IE). Las banderas de interrupción son activadas independientemente de si la interrupción esta habilitada o no.

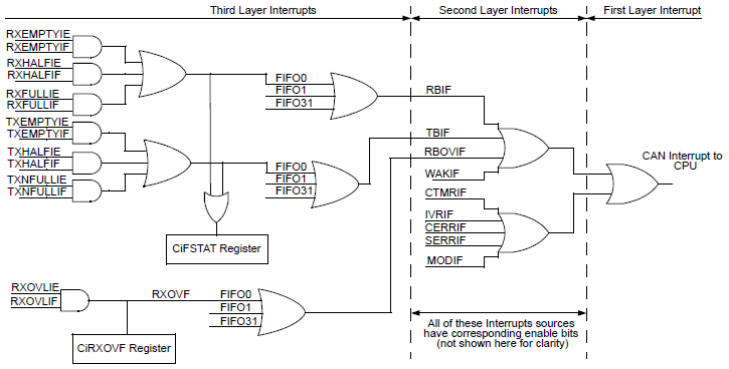


Figura 5.5: Multiplexación de interrupciones CAN.

Las interrupciones de la segunda capa son nueve:

- ***TBIF:*** Evento en FIFO de transmisión.
- ***RBIF:*** Evento en FIFO de recepción.
- ***CTMRIF:*** Evento del timer timestamp.
- ***MODIF:*** Evento por cambio de modo de operación en el módulo.
- ***RBOVIF:*** Evento por overflow de alguna FIFO.
- ***SERRIF:*** Interrupción por error en el módulo CAN.
- ***CERRIF:*** Interrupción por error en el bus.
- ***WAKIF:*** Interrupción de wake-up.
- ***IVRIF:*** Recepción de un mensaje inválido.

Como se puede ver, TBIF, RBIF, RBOVIF, CERRIF y SERRIF tienen múltiples fuentes de interrupción cada una. Las interrupciones de la tercera capa reflejan el estado de las FIFO de transmisión y recepción, estas son:

- ***TXNFULLIF:*** FIFO de transmisión no esta llena.
- ***TXNHALFIF:*** No se ha llenado media FIFO de transmisión.
- ***TXNEMPTYIF:*** FIFO de transmisión no esta vacía.
- ***RXFULLIF:*** FIFO de recepción llena.
- ***RXHALFIF:*** FIFO de recepción llena hasta la mitad.
- ***RXEMTYIF:*** FIFO de recepción no vacía.
- ***RXOVLIF:*** Overflow en FIFO de recepción.

La configuración de la tasa de envío de bits se calcula a partir del tiempo de bit CAN.

$$f_{BAUD} = \frac{1}{t_{bit}}$$

Ecuaciones 5.1: Cálculo de tasa de envío CAN.

Este tiempo de bit, esta dividido en cuatro segmentos para compensar desviaciones de fase debido las diferencias entre osciladores de distintos módulos, tiempos de propagación, etc. Estos segmentos nos se solapan unos con otros, están divididos en cuantos de tiempo (Time Quantum Tq). Un Tq es una unidad de tiempo derivada de la frecuencia del oscilador. El número total de Tq en un tiempo de bit se programa entre 8 y 25.

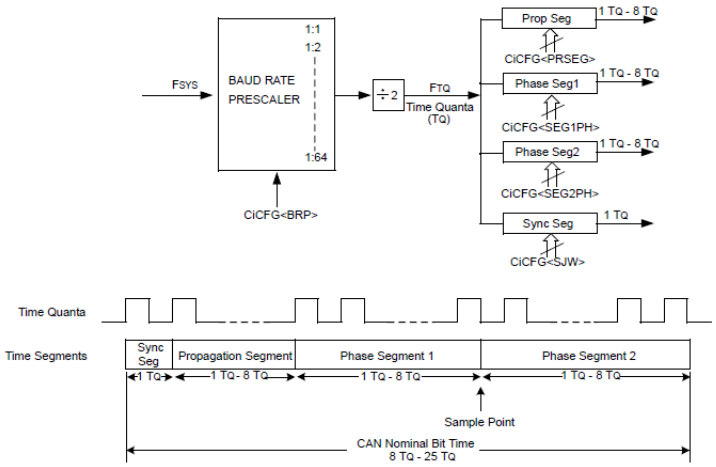


Figura 5.6: Tiempo de bit CAN.

La obtención del Tq se puede ver en la figura anterior, se ve también que se han de configurar unos registros del módulo para obtener el número de Tq en los diferentes segmentos de tiempo. Estos segmentos de tiempo son:

- ***Segmento de sincronización:*** Este segmento de tiempo sincroniza los diferentes nodos conectados al bus. En este tiempo se espera un flanco de bit, acorde con el protocolo CAN. Este tiempo es fijo de 1Tq.
- ***Segmento de propagación:*** El segmento compensa retardos que puedan haberse producido en el bus o el producido por los distintos transcivers conectados al bus. El número de Tq de este segmento de tiempo varia entre 1 y 8.

- ***Segmento fase1:*** Este segmento compensa errores que puedan ocurrir en la sincronización de flancos entre nodos, este segmento se puede alargar durante la resincornización interna. Su duración varia entre 1 y 8 Tq.
- ***Segmento fase2:*** Este segmento compensa errores que puedan ocurrir en la sincronización de flancos entre nodos, este segmento se puede acortar durante la resincornización interna. Su duración varia entre 1 y 8 Tq.
- ***Segmento de salto:*** Este segmento es el "salto" máximo de tiempo que se hace durante la sincronización del punto de muestreo.

El cálculo del Time Quantum se realiza con las siguientes ecuaciones primero se calcula la frecuencia del Tq, se obtiene el valor del prescaler (BRP, Baud Rate Prescaler) y se selecciona el número de Tq asignados a cada segmento:

$$f_{TQ} = N * f_{BAUD}$$

$$BRP = \frac{f_{SYS}}{2 * f_{TQ}} - 1$$

$$t_{bit} = t_{sync} + t_{prop} + t_{PH1} + t_{PH2}$$

Ecuaciones 5.2: Cálculo del Tq.

Para el desarrollo de la programación se han empleado las funciones del fichero de cabecera CAN.h de Microchip.

5.7.1. El VAI en el Bus CAN UPM-006.

Dentro del enunciado del proyecto el sistema de admisión variable se indica que debe conectarse a un bus CAN en el monoplaza, es decir, el sistema va a ser un módulo del bus CAN. En automoción a estos módulos conectados al bus CAN se los conoce como calculadores.

No se dispone o no hay información alguna a cerca del formato en que se han de transmitir los datos a los posibles receptores. Para evitar la carga del bus con demasiadas tramas de un único nodo transmitiendo y ocupando el bus repetidamente para transmitir la distinta información que disponga y ya que CAN permite la transmisión de hasta 8 octetos en una trama, se ha creado un tipo de dato para que se envíe toda la información referente al "calculador VAI" (Variable Air Intake) en una única trama. La descripción de estos tipos de datos se realiza en los siguientes apartados.

Para el diseño y la configuración de las comunicaciones hacen falta saber qué tipo y número identificador se asigna a los mensajes enviados por el módulo VAI, tasa de bits, qué más módulos hay en el bus, qué información envían y qué información se debe introducir al bus.

- Para la realización de este software se va a fijar una tasa de bits de 125kbps, el tipo de trama será estándar (SID), los datos a enviar serán las rpm, el TPS medidos y la distancia a la que se encuentran los "runners" respecto a la posición de reposo.
- Del mismo modo configurado, para introducir tablas de posiciones debe haber algún módulo que realice esta tarea (usualmente suelen ser portátiles con un software apropiado y conexión al bus CAN); este módulo se conecta al bus para configurar algún calculador o extraer alguna información y luego se desconecta.
- La información enviada por el VAI debe ser utilizada por algún otro módulo, por ello, puede haber algún nodo monitor que guarde un registro de la evolución de las rpm y el TPH o las transmita hacia el pit-lane de alguna manera.
- También esta información debe ir al piloto, por lo que debe haber otro módulo de control en el bus que haga de interfaz con el piloto, este hecho se aprovecha para que el módulo VAI reciba distintas órdenes que pueda realizar el piloto desde dicho módulo.

Según todo esto la situación del bus es la de la figura de abajo.

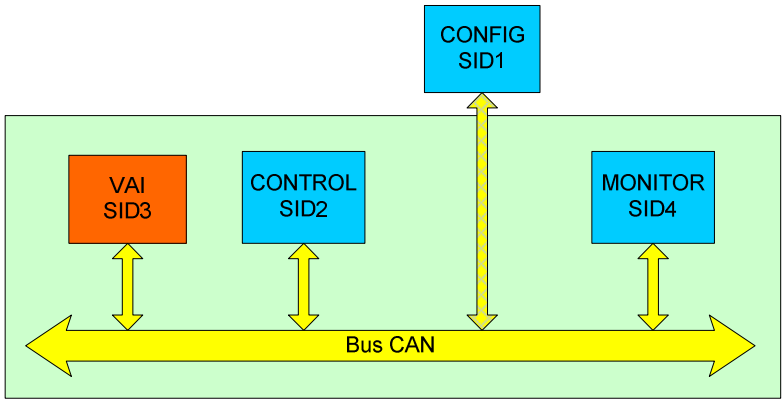


Figura 5.7: Esquema Bus CAN UPM-006.

Partiendo de este esquema se van a configurar 3 FIFOs, una de transmisión y dos de recepción y se van a aceptar solo los mensajes enviados por el módulo de configuración y el módulo de control. Estos van a enviar las órdenes hacia el módulo VAI, otra orden enviada no produce ningún efecto en el calculador y se van a configurar dos filtros de aceptación por lo que cualquier otro mensaje con SID distinto al de los filtros no es aceptado.

- ***Módulo CONFIG:*** Su SID es 0x001, envía las órdenes para crear una tabla nueva, modificar una posición y cambiar el valor de %TPS mínimo.
- ***Módulo CONTROL:*** Su SID es 0x002, envía las órdenes de apagado/encendido de la actuación sobre los "runners", reset del calculador y cambio de tabla o mapa de motor.

5.7.2. Configuración del módulo CAN VAI.

Se ha configurado una FIFO para la transmisión, con un tamaño de 10 mensajes, sin respuesta a trama remota; dos FIFOS de recepción; y dos filtros, uno para aceptar mensajes SID1 provenientes del módulo de configuración que se guardaran en la FIFO1 y otro para los mensajes con SID2 del módulo de control que se guardan en la FIFO2, cada una de estas FIFOs tiene un tamaño de 10 mensajes.

Los eventos que permitirán la señalización de una interrupción en el microcontrolador habilitados son: RBIF, RBOVIF, SERRIF, CERRIF, de la segunda capa. Y de la tercera capa se habilitan: RXEMTYIF y RXOVLIF de cada una de las FIFO de recepción. Lo que hace la rutina de atención a la interrupción es identificar la fuente de la interrupción y actuar sobre cada una de estas. Cuando se detecta errores en el módulo (SERRIF) el fabricante aconseja para una de estas fuentes de error el reinicio del módulo, para el resto de fuentes de error se notificara por CAN esta situación ya que no se especifica nada. Si lo que se detecta es error en el bus (CERRIF) se notifica externamente por el puerto E el estado de error del nodo (activo, pasivo, desconectado). El diagrama de flujo de la interrupción es como sigue:

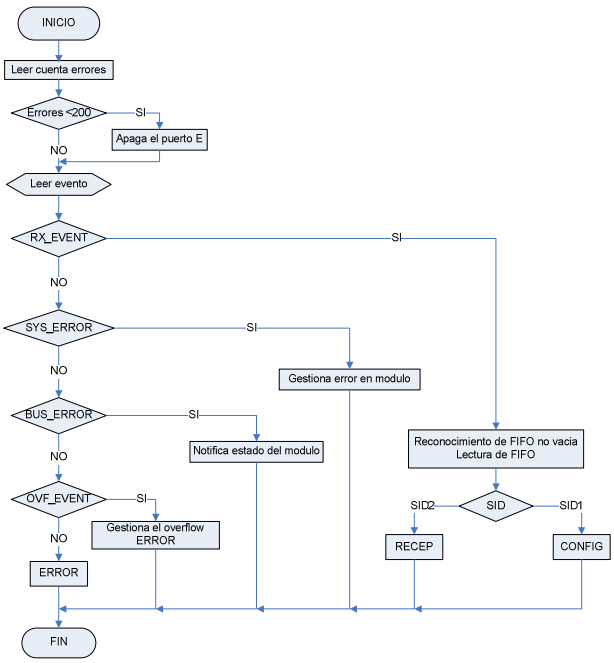


Figura 5.8: Diagrama de flujo de la interrupción CAN.

El fabricante recomienda que el tiempo Tq y los tiempos de cada fase se diseñen de tal forma que el punto de muestreo de bit, que se realiza entre la fase 1 y la 2 quede al 70% del tiempo de bit CAN. Se ha optado que haya 10 Tq (1 sinc + 3 prop + 3 fase1 + 3 fase2) y 1 Tq para la sincronización interna.

En el ANEXO II se ve la codificación de la configuración del módulo CAN en la función CAN_config(). Para la realización de esta configuración se han empleado las funciones de cabecera de CAN.h, que una vez comprendidas facilitan el manejo del módulo.

5.7.3. Transmisión CAN.

El envío de tramas se realiza dando la orden de envío, es decir, el envío no se hace por interrupción en la FIFO de transmisión. El envío de tramas se realiza empleando función CAN_transmit(DATOTX *dato). Su código se puede ver en el ANEXOII y su diagrama de flujo es como se indica a continuación:

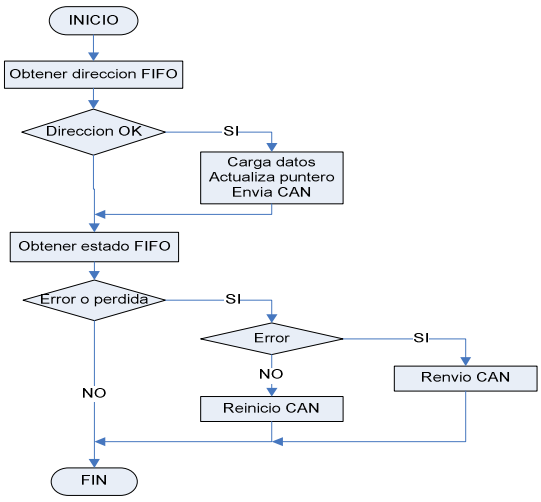


Figura 5.9: Diagrama de flujo de recepción CAN.

En la estructura empleada para enviar datos (DATOTX) se distribuye la información de todos los datos a enviar en los 64 bits de datos que se pueden transmitir en una única trama. Estos datos enviados son:

- ***RPM:*** Es un entero de 16 bits que emplea los bits menos significativos de los datos, para enviar las revoluciones por minuto del motor de combustión.
- ***TPH:*** Es el % de TPH aplicado, es un entero de 8 bits.
- ***mm:*** Es la distancia en mm que hay desde la posición de reposo de los "runners" y la posición actual del motor paso a paso. Es un entero de 8 bits.

- ***n_mapa:*** Un entero de 4 bits que envía el número de mapa motor en uso.
- ***Status:*** Es un dato de 4 bits que junto con el estado dan información de la situación interna del módulo.
- ***Estado:*** Es un entero de 4 bits que junto con Status se emplean para obtener el estado interno del módulo.

DATOTX								
RPM	TPH	mm	n_mapa	vacio	extra	vacio	orden	vacio

Tabla 5.2: Estructura de datos en la trama de transmisión VAI.

5.7.4. recepción CAN.

La recepción de mensajes CAN se realiza empleando la interrupción de FIFO no vacía, que permanecerá a '1' mientras no se vacíe la FIFO que produjo la interrupción. En la rutina de atención a la interrupción por CAN, primero se identificará si la fuente de la interrupción es por recepción de un mensaje, identificando, también, qué FIFO es la que se debe atender, a continuación se lee el mensaje de la FIFO y se indica al programa principal que tiene que procesar un nuevo dato. Para realizar parte de esta tarea se emplea la función CAN_EVENT_CODE CAN_recive(DATORX *dato, CAN_MODULE_EVENT modEvent) que se le pasa, el evento sucedido en el módulo así como el puntero del buffer de recepción, que es un estructura que se crea para la recepción de los datos de las tramas recibidas. La función devuelve un tipo definido en CAN.h que es el número de filtro señalado por el mensaje recibido, identificando al SID del mensaje. Identificando de este modo, además, qué módulo ha sido el que ha transmitido el mensaje. Se ofrece a continuación el diagrama de flujo de esta función que es el de la figura de abajo, la codificación está en el ANEXOII.

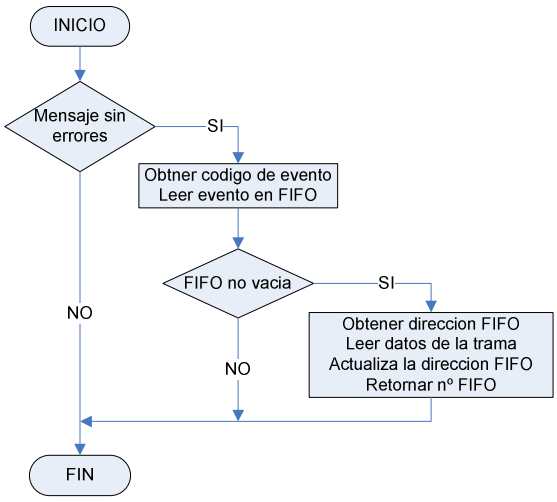


Figura 5.10: Diagrama de flujo de CAN_recive.

La estructura de los datos se muestra en la tabla de abajo (DATORX), que es el formato del los datos recibidos. Para que la comunicación funcione los datos se deben recibir de la forma adecuada, siendo necesaria la situación de estos campos de datos de una forma fija por parte de los transmisores. Debido a lo cual se ofrece, también, cómo son los datos en estas tramas SID1 y SID2:

DATORX					
RPM	mm	MAXPOSC	VACIO	extra	orden
SID1					
RPM	mm	MAXPOSC	NO IMPORTA	extra	orden
SID2					
NO IMPORTA				GEAR	orden

Tabla 5.3: Estructura de los datos en la recepción CAN.

Siendo los bits menos significativos los correspondientes a los datos del lado izquierdo de la tabla. Se dejan espacios por si en algún futuro se amplían las órdenes o se añade alguna opción adicional al módulo.

La descripción de cada uno de los datos recibidos es la siguiente:

- ***RPM:*** Es un entero de 16 bits que emplea los bits menos significativos de los datos, recibe el umbral máximo de rpm para una posición en mm de una tabla de mapa motor.
- ***mm:*** Es la distancia en mm que hay desde la posición de reposo de los "runners" y la posición del motor paso a paso mientras no se supere el umbral de rpm. Es un entero de 8 bits.
- ***MAXPOSC:*** Un entero de 8 bits que indica el número máximo de posiciones al crear una tabla nueva.
- ***Extra:*** Es un dato de 8 bits que contiene información adicional que algunas órdenes necesitan (número de mapa motor o % de nueva posición de TPS).
- ***Orden:*** Es un entero de 8 bits que recibe la orden que se ha recibido por el bus CAN.

5.7.5. CANStatus.

Es una variable de un tipo definido de estructura, CANst, que contiene dos tipos de enteros, que se presentan a continuación:

- ***Info:*** Es un entero que emplea el módulo para contestar cuando esta en modo configuración. Un 3 indica orden realizada, un 2 notifica un error en la posición enviada al crear la tabla, un 1 indica tabla en creación y que la posición enviada es correcta y 0 es error al realizar la orden.
- ***Ovf:*** Un '1' indica que se produjo overflow en alguna de las FIFO de recepción.

Esta variable se envía al bus CAN cuando el módulo está en el estado de configuración. Y es la que emplea el calculador VAI para notificar su respuesta ante las órdenes.

5.8. Adquisición TPS.

La adquisición del TPS se realiza empleando el ADC-10bits ya que se va a leer un valor de tensión proporcional al porcentaje de actuación del acelerador. El ADC del PIC32MX tiene las siguientes características, y su diagrama de bloques se ve en la figura 5.11:

- La conversión se realiza por aproximaciones sucesivas.
- Velocidad de conversión hasta 1Msps.
- Hasta 16 entradas analógicas.
- Tensión de referencia externa para la conversión.
- Un amplificador diferencial unipolar Sample-and-Hold.
- Escaneo automático de canales de entrada.
- Disparador de conversión seleccionable.
- Un buffer de conversión de 16 WORD.
- Selección del modo de llenado del buffer.
- Ocho tipos de datos para el resultado de la conversión.
- Operaciones en bajo consumo

Las entradas analógicas, AN0-AN15, están conectadas a través de multiplexores a un SHA. Los multiplexores de entrada pueden intercambiarse entre conversiones. Únicamente se pueden realizar conversiones de señales diferenciales unipolares. En el modo de escaneo se convierten los canales del MUXA seleccionados por la configuración. El resultado de la conversión, según el modo de llenado va hacia el buffer y al ser leído, el resultado se convierte en uno de sus 8 formatos de datos de 32bit de salida.

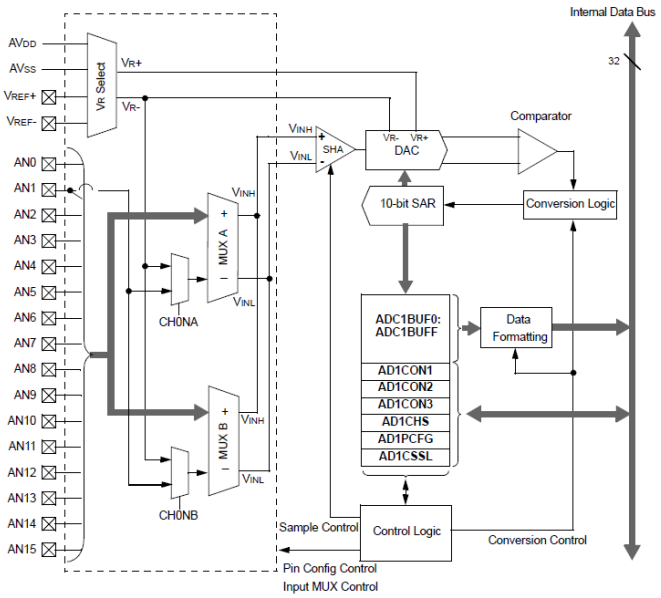


Figura 5.11: Diagrama de bloques del ADC.

En la configuración del ADC para la adquisición del TPS el tipo de dato de salida del buffer es el formato entero de 16-bits, rellenándose con '0' hasta 32 bits. El inicio de la conversión es manual y el muestreo automático. La tensión de referencia se obtiene de los pines para la tensión de referencia externa, solo se realiza un muestro/conversión para señalizar el final de la conversión, el buffer es único de 16-WORD, la entrada analógica positiva se hace por el MUXA en el pin AN0, y la entrada negativa seleccionada es la tensión de referencia negativa (Vref-). El periodo de adquisición (Tad) se configura a partir de la frecuencia reloj de los periféricos (PBCLK).

$$T_{AD} = 2 * T_{PB}(ADCS + 1)$$

$$ADCS = \left(\frac{T_{AD}}{2 * T_{PB}} \right) - 1$$

Ecuaciones 5.3: Cálculo del periodo de adquisición.

ADCS son los bits 0..7 del registro AD1CON3. Como el ADC es de 10 bit la sensibilidad del ADC viene dada por el tamaño de un LSb, mucho menor que el incremento mínimo de tensión que se quiere medir para un incremento del 1% en la posición del TPS, que es de 0.033 V.

$$1LSb = \frac{\Delta Vref}{2^{10}} = 0.003V$$

Ecuaciones 5.4: Cálculo de 1 LSb.

Con el código digital (CD) leído del buffer se calcula su valor de tensión acondicionada, con ella se calcula el %TPS obteniendo el tanto por cien de la tensión leída respecto a la tensión de referencia. El código del ADC y la configuración del mismo se ven en el ANEXO II.

$$Vacond = CD\frac{\Delta Vref}{2^{10}}$$

$$\%TPS = Vacond\frac{100}{\Delta Vref}$$

Ecuaciones 5.5: Cálculo del %TPS.

5.9. Adquisición RPM.

Debido a que el sensor con el que capta las revoluciones del motor del monoplaza genera una señal variable en frecuencia, en función de la velocidad de giro del motor. Para medir esta frecuencia se emplea el Input Capture 1 (IC1). Este módulo en el PIC32MX tiene varios modos de operación, con los que identifica el tipo de evento en el pin a capturar, cuando ocurre una evento en su pin de entrada se captura el valor del registro TMR de un timer que se almacena en un buffer. El uso del input capture es muy apropiado para aplicaciones en las que se necesite medir un periodo.

Teniendo en cuenta que es según el modo de operación como se genera la captura del timer, a continuación se exponen los modos de operación existentes:

- Captura el timer en cada flanco de subida aplicado al pin del IC.
- Captura el timer en cada flanco de bajada aplicado al pin del IC.
- Captura el timer en cada 4º flanco de subida aplicado al pin del IC.
- Captura el timer en cada 16º flanco de subida aplicado al pin del IC.
- Captura el timer en cada flanco de subida y flanco de bajada aplicado al pin del IC.
- Captura el timer a partir de un flanco especificado y cada flanco siguiente aplicado al pin del IC.

Como el buffer es una FIFO de cuatro niveles se pueden capturar hasta cuatro eventos antes de generar una interrupción, también tiene un prescaler en la entrada. Su diagrama de bloques es el siguiente.

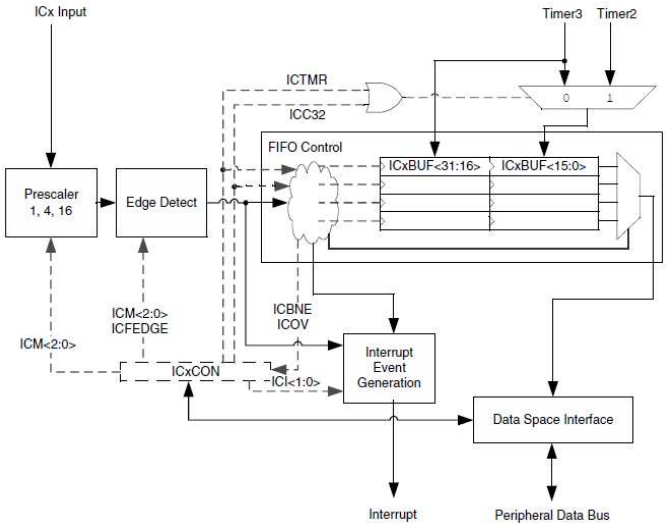


Figura 5.12: Diagrama de bloques de IC1.

El timer 3 es el que se dedica para el IC1 y el modo de operación del IC1 empleado es realizar una captura por cada flanco detectado, generando una interrupción por cada flanco detectado. Este modo de operación es aconsejado para su uso con sensores de efecto Hall como el que se esta empleando en este caso. Este modo tiene la particularidad que genera una interrupción por cada flanco detectado, ignorando la configuración del número de eventos que se hubiera puesto para generarla.

El timer3, se ha configurado para que funcione a la frecuencia de reloj de periféricos, con las capturas del IC1 realizadas se obtiene el número de cuentas que ha hecho el timer y a partir de este número se obtiene la frecuencia capturada. Y dividiendo esta frecuencia capturada entre la sensibilidad del ATS616 + engranaje, que es 0.5Hz/rpm, se obtienen las rpm del motor de combustión. La codificación de configuración y de la obtención de rpm está en el ANEXO II. El cálculo se realiza con las siguientes ecuaciones.

$$Cuenta = CapturaFlanco2 - CapturaFlanco1$$

$$f_{capturada} = \frac{1}{Cuenta * T_{PCLK}}$$

$$rpm = \frac{f_{capturada}}{S_{ATS}}$$

Ecuaciones 5.6: Cálculo de rpm con IC1.

Esta forma de captura hace que el error en esta medida sea mayor cuando el periodo a medir sea menor, es decir, a frecuencia de 7500 Hz equivalentes a 15000 rpm del motor de combustión. Sabiendo el periodo mínimo de la señal a medir, se obtiene el número de pasos que debe haber

en la cuenta realizada por el timer. La variación en la medida es debida a que en algunas ocasiones el valor de cuenta del timer valdrá uno más o uno menos del valor ideal capturado. Además, al tener que dividir entre la sensibilidad del ATS616 + engranaje (0.5Hz/rpm), el error aumenta por el redondeo interno de esta operación. Con las siguientes ecuaciones se obtiene la desviación máxima en la medida de rpm.

$$f_{\max} \rightarrow T_{\min} = 133.33us$$
$$Cuenta = \frac{T_{\min}}{T_{PCLK}} = 1333$$
$$f_{capturada} = \frac{1}{S_{ATS}} \frac{1}{Cuenta * T_{PCLK}} (rpm)$$
$$Cuenta1 = 1332 \rightarrow f_{capturada} = 15015(rpm)$$
$$Cuenta2 = 1334 \rightarrow f_{capturada} = 14992(rpm)$$
$$Error = \frac{\pm(Cuenta1 - Cuenta2)}{2} = \pm 11.5(rpm)$$
$$f_{capturada} = f_{RPM\_motor} \pm 11.5(rpm)$$

Ecuaciones 5.7: Error en la medida con el IC.

5.10. Control del motor.

Se llama control motor al conjunto de periféricos y código que se encargan de captar el recorrido del motor y mantenerlo bajo un control de recorrido y velocidad. Como se ha comentado, el control motor necesita de dos timer, un puerto, un PWM con detección de fallo y una interrupción externa. El control motor maneja la distancia actual y la posición en la tabla del motor paso a paso, genera el perfil de velocidad, detecta fallos en el DRV8805, y si el actuador estuviera en movimiento y fuera necesario, cambia el sentido del desplazamiento.

5.10.1. Actuación sobre DRV8805.

Para enviar las señales de control de dirección, enable, modo de paso y reset se emplea el puerto D (PORTD). Para facilitar el manejo de este puerto y la actuación sobre el driver, se ha desarrollado una función BOOL ControlMotor(int estado) que se encarga de actuar sobre el puerto para configurar y utilizar el DRV8805. Esta función se emplea a lo largo de la función principal, actuando sobre el puerto E y consiguiendo que el motor se mueva de forma controlada, siendo todo este control del driver transparente al programa principal.

Entonces desde el programa principal se indica al motor que ha de moverse mediante el empleo de dicha función, luego todos los periféricos del control de motor captan y generan la información necesaria para llevar el motor a la distancia acorde a lo que indique la tabla de posiciones del mapa de motor. El diagrama de flujo de esta función ControlMotor(estado) es:

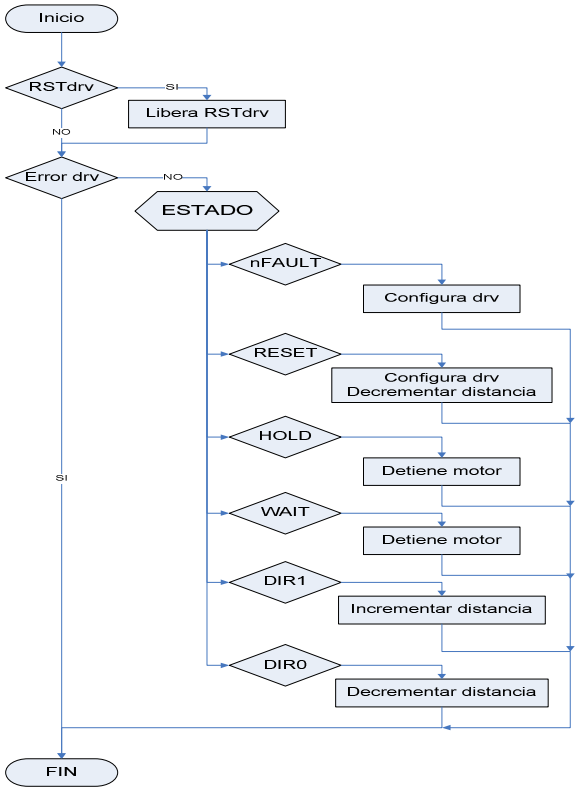


Figura 5.13: Diagrama de flujo de ControlMotor(estado).

El modo de funcionamiento del DRV8805 es full-step. La función comprueba si se reiniciaron las cuentas internas del driver, y si no se encuentra error en el mismo, se configura el puerto con la orden adecuada, se actúa sobre el PWM y se inicia el perfil, según la orden recibida. Retorna un valor lógico “FALSE” en todos los casos, excepto cuando se recupera de un estado de sobre temperatura o corriente en el driver, que retorna “TRUE”. La codificación de esta función esta en el ANEXO II.

5.10.2. Adquisición SFC.

Físicamente el sensor de fin de carrera es un pulsador, entonces se precisa de una interrupción que reconozca estos cambios de tensión, para ello se emplea la interrupción externa 3 (EXT3). El microcontrolador tiene 5 interrupciones externas, estas son sensibles a los flancos de subida o de bajada para generar una interrupción.

La rutina de atención a esta interrupción lo que hace es cambiar la dirección de desplazamiento y forzar la búsqueda de su posición de reposo, si el motor estaba bajando. Y en caso de que el paso a paso estuviera subiendo, se busca la posición más alta de la tabla. La posición de reposo son 2 mm de separación con este sensor. Por bus CAN se comunica esta situación indicándose con el estado del módulo cuando ha terminado de posicionar los "runners" en la posición de reposo. El sensor de final de carrera solo es sensible mientras el motor se esta desplazando, si por algún motivo este pulsador es accionado en cualquier otro estado no se produce ninguna acción.

Este margen de 2 mm se puede variar por codificación software cambiando una variable, esto no se permite como opción de configuración por CAN, ya que este offset es un ajuste de posición sobre la distancia entre los sensores de final de carrera y el recorrido que hay que realizar, en el espacio habilitado para los "runners". En el caso de un margen de 2 milímetros, se supone que se dispone de un espacio total de recorrido de 184 mm. Tras un calibrado el motor estaría disminuyendo la distancia, hasta presionar este sensor, la dirección cambia y se para al recorrer este offset de posición de reposo.

5.10.3. Adquisición nHOME.

Esta es una señal generada por el DRV8805, que se emplea como realimentación de la distancia avanzada por el motor, seria una señal equivalente a la que generaría un ENCODER. Para su captación se emplea el timer 4. La familia PIC32 ofrece dos tipos de timers útiles para generar bases de tiempo estables o cuenta de pulsos externos, que es como se empleará este timer.

Los dos tipos de módulos timer tienen en común las siguientes características:

- 16-bit timer/counter
- Selección software interna o externa de la fuente de reloj.
- Generación de la interrupción programable y prioridad
- Puerta externa de cuenta de pulsos.

Además de estas características comunes cada tipo de timer tiene las suyas propias

- ***El tipo A:*** Opera en modos de bajo consumo, dispone de prescalers seleccionables por software y la posibilidad de timer/counter asíncrono.

- ***El tipo B:*** Puede formar un 32-bit timer/counter, prescalers seleccionables y eventos de disparo seleccionable. Como el ADC.

En la siguiente figura se presenta el diagrama de bloques de un timer tipo B, como es el timer 4.

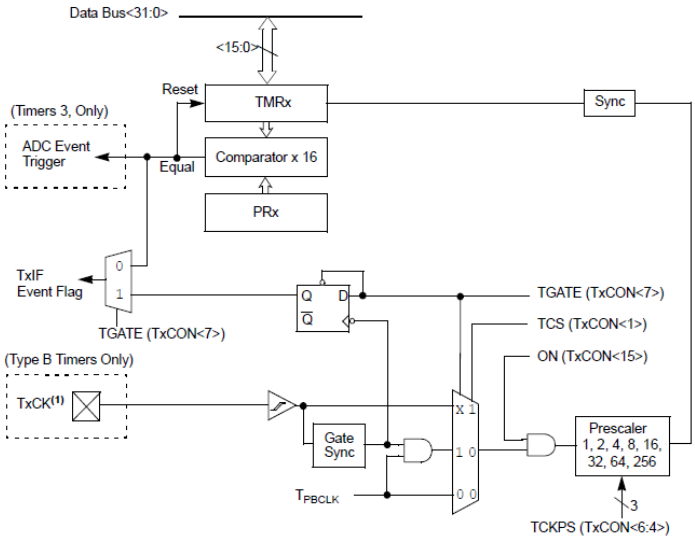


Figura 5.14: Diagrama de bloques del timer tipo B.

Según el catálogo del driver y como se ve en la tabla 4.3, la señal nHOME proveniente del driver genera un flanco de bajada con el cuarto paso realizado. Según esto y las características del motor, se genera un pulso cada desplazamiento de 0.1mm. Por ello, para controlar la resolución del desplazamiento en el perfil de velocidad, se genera la interrupción del timer 4 cuando se haya recorrido 1 mm, lo cual ocurre cuando se cuentan diez flancos de bajada de la señal nHOME; por esto se configura el registro de comparación PR4 = 9 para generar la interrupción.

La rutina de atención de la interrupción del timer 4 es la que controla la distancia actual del motor paso a paso respecto a su posición de reposo. Con lo que controla cuándo debe detenerse el motor al alcanzar su posición adecuada. Cambiar el sentido de desplazamiento de los “runners” si fuera necesario. Y es la salida del estado de RESET cuando el motor paso a paso alcanza su posición de 0 mm a 8000 rpm, que es la posición minima obligatoria en cualquier tabla. Es decir, controla que el motor paso a paso tras el calibrado o el reset se detenga a la distancia offset del SFC, iniciando sus valores de distancia. Para el manejo del DRV8805 esta rutina se sirve de la función ControlMotor() vista anteriormente. El diagrama

de flujo se ve en la siguiente figura, la configuración y la programación de este periférico esta en el ANEXO II.

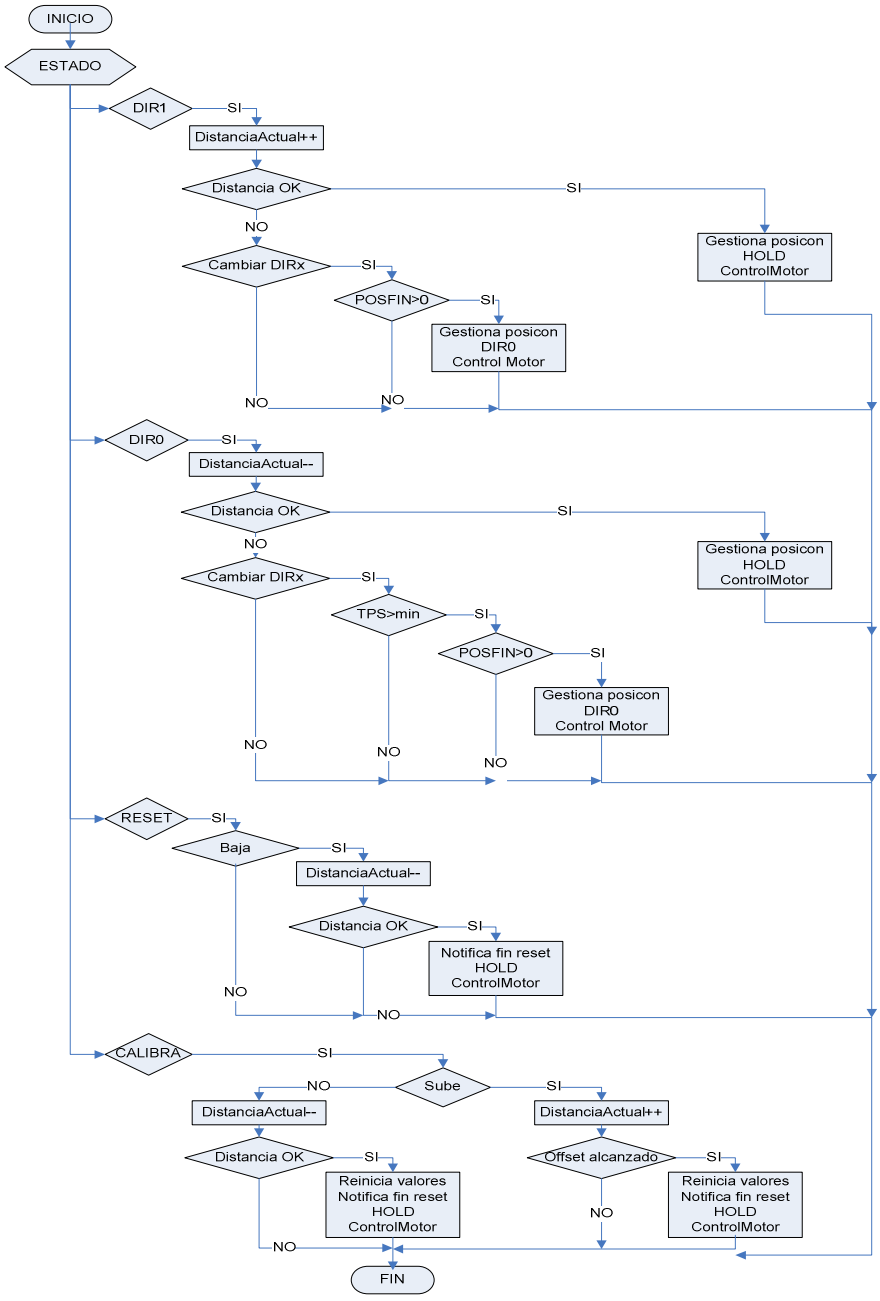


Figura 5.15: Diagrama de flujo de la interrupción del timer 4.

5.10.4. Perfil de velocidad.

El perfil de velocidad se genera empleando el Output Compare (OC5) para producir la señal de pulsos variables en frecuencia. El Output Compare se usa para generar un pulso o un tren de pulsos en respuesta a la selección de base de tiempo.

Estas son algunas de las características de este periférico:

- Múltiples salidas OC en un mismo dispositivo
- Modos de comparación simple o doble.
- Generación de pulsos simples y continuados.
- Modulación por ancho de pulso (PWM).
- Interrupción programable en el evento de comparación.
- Detección de PWM Fault con desactivación automática de salida.
- Base de tiempo de 16 o 32 bits programable.

Su diagrama de bloques es el siguiente:

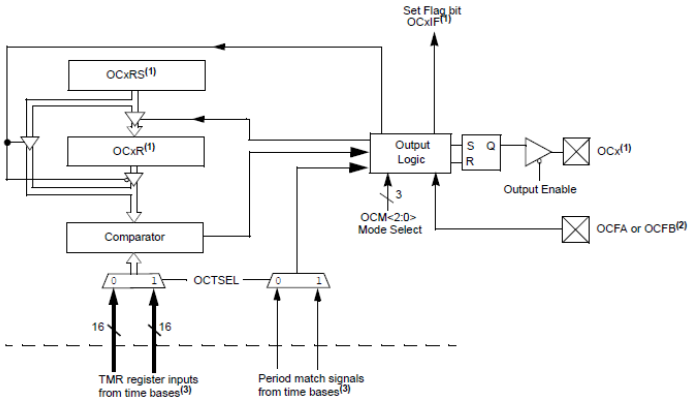


Figura 5.16: Diagrama de bloques de Output Compare.

El modo de operación empleado es PWM, y la base de tiempo elegida es la del timer 2. Con la programación del PR2 de este timer se obtiene el periodo, y cargando la mitad del periodo cargado en PR2 en el registro de actualización de ciclo de trabajo del PWM. Para su codificación, se crea un array de datos en los que figuran los distintos periodos que se cargan en PR2. Estos periodos se obtienen de las frecuencias calculadas en la obtención del perfil de velocidad del motor (Apartado 4.4.1), con la configuración del prescaler del timer 2 a N = 1, la fuente de reloj es la del bus de periféricos. Se configura este timer tipo B como 16 bit timer.

$$T = \frac{1}{f}$$

$$PR2 = \frac{T_{i\_perfil}}{T_{PBCLK} * N}$$

$$OC5RS = \frac{PR2}{2}$$

Ecuaciones 5.8: Registros del perfil de velocidad.

Con estas ecuaciones y el número de pulsos que se deben enviar en cada una de las tres frecuencias se crea un array de datos de 40 enteros, uno por paso con los valores que se cargan en el periodo del timer 2. En la rutina de atención a la interrupción de este timer se cargan los nuevos valores de periodo y de ciclo de trabajo. Los tres valores obtenidos para las frecuencias calculadas en la obtención del perfil de velocidad son:

$$f_1 = 200(pasos/s) \rightarrow PR2 = 0xC350$$
$$f_2 = 250(pasos/s) \rightarrow PR2 = 0x9C40$$
$$f_3 = 275(pasos/s) \rightarrow PR2 = 0x8E0B$$

Ecuaciones 5.9: Valores del registro PR2.

5.10.5. RSTStatus.

Es una variable de un tipo creado para proporcionar más información sobre el estado del VAI al bus CAN, en realidad, es una estructura de 4 bits, a los que se les puede asignar valor de forma independiente. Además se emplean para la realización del reset del módulo VAI de una forma controlada. Los 4 bits de que consta son estos:

- *RSTMotor:* Un '1' indica que se va a realizar el reset de posicionado del motor paso a paso.
- *RSTTabla:* Un '1' indica que se van a resetear la tabla y el TPS mínimo.
- *RSTMedidas:* Un '1' indica que se van a reiniciar algunos periféricos empleados.
- *RST:* Un '1' indica que se esta moviendo el motor buscando la posición de offset o reposo de los "runners".

6. Desarrollo del sistema.

6.1. Microchip Bus Monitor.

Con un hardware y un software ya establecidos, se va a evaluar la respuesta del calculador en las mediciones de rpm y TPS del monoplaza, este resultado es transmitido por el módulo VAI al bus CAN. Además, como el calculador recibe órdenes, de los otros nodos por el bus CAN, para poder definir al instrumento es necesario saber qué datos se están enviando al bus CAN. Por otra parte con el fin de poder comprobar el funcionamiento del nodo VAI, cuando recibe una orden, se emplea el kit de desarrollo de microchip MCPC515 CAN Bus Monitor Demo Board.

Este kit consta de dos placas idénticas, que se pueden conectar entre sí para crear un bus CAN sencillo de dos nodos que puede ser controlado y monitorizado por un PC empleando el software de microchip, mediante la conexión de una de ellas al PC por USB. A este bus se le pueden conectar más módulos incrementando el número de nodos del bus.

Los dos módulos son iguales en todo, su función esta determinada por la conexión de una de ellas al PC. El nodo conectado al PC es el nodo monitor, el otro e nodo generador de trafico, siendo indiferente cuál de ellos se conecte al PC. Si se conecta una de las PCBs a un bus ya existente y esta tarjeta al PC, empleando el software se puede monitorizar este bus, el software, permite la configuración de varios registros del MCP2515, así como la transmisión y recepción de mensajes.

El software de microchip consta de cinco ventanas.

- ***Ventana de estadísticas del bus:*** Muestra el estado del bus incluyendo el encendido/apagado del bus, la carga de tráfico que está soportando, el número de mensajes transmitidos y recibidos y el estado de la temporización de bit.

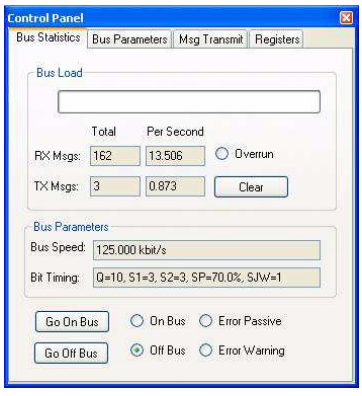


Figura 6.1: Ventana de estadísticas MCP2515.

- ***Ventana de parámetros del bus:*** Se usa para configurar el tiempo de bit, el punto de muestreo y el salto de sincronización.

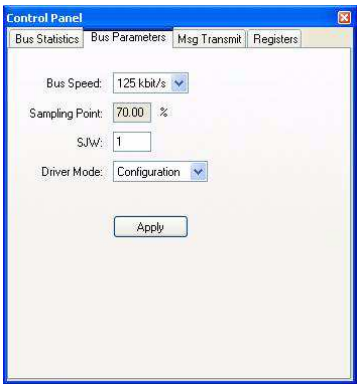


Figura 6.2: Ventana parámetros del bus MCP2515.

- ***Ventana de transmisión:*** Esta pantalla se emplea para configurar y transmitir mensajes. La configuración del mensaje permite seleccionar si la trama será extendida o estándar, dar un número de identificador de mensaje, la longitud del los datos y los datos introducidos.

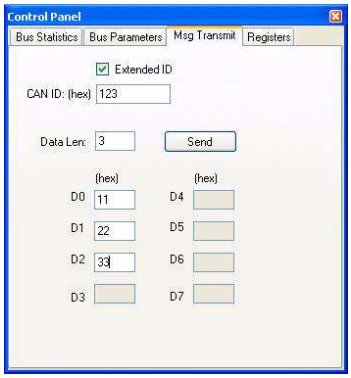


Figura 6.3: Ventana de transmisión MCP2515.

- ***Ventana de salida:*** Esta ventana muestra los mensajes transmitidos y recibidos por el MCP2515.

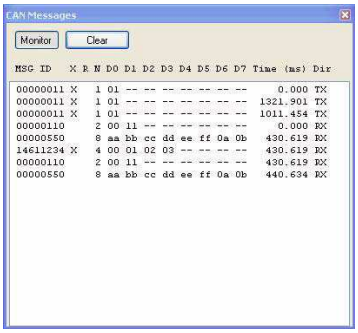


Figura 6.4: Ventana de salida MCP2515.

- ***Ventana de configuración del MCP2515:*** Esta ventana permite acceso a los registros del MCP2515.

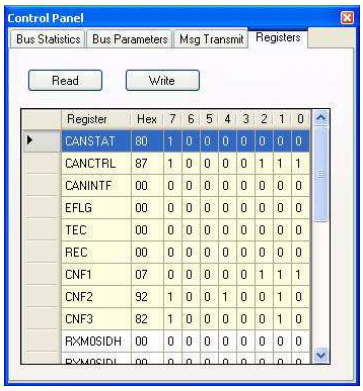


Figura 6.5: Ventana configuración MCP2515.

6.1.1. Datos del VAI en el bus CAN.

Los datos transmitidos por el módulo VAI se cargan empleando un buffer de envío de datos del tipo DATOTX. Al ser mostrados en la ventana de salida se ven de la siguiente manera:

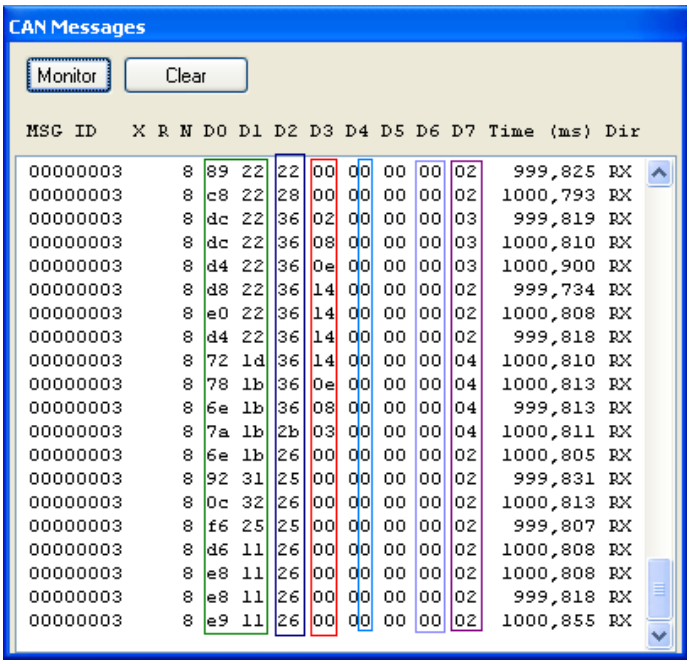


Figura 6.6: Datos del módulo VAI.

En la primera columna se ve el número de identificador de mensaje que envía o recibe el mensaje, en este caso es 0x003, que es el SID asignado a los mensajes del módulo VAI. En la

siguiente columna (N) se ve el número de octetos transmitidos. Los datos están distribuidos por octetos del D0 al D7.

En verde se ven las rpm medidas, en azul oscuro la posición del TPS, en rojo la distancia del los "runners" a la posición de reposo, los 4 bits menos significativos del octeto D4 en azul claro, corresponde al número de mapa motor en uso; en morado claro se envía RSTStaus o CANStatus y, finalmente, en morado oscuro el estado del módulo VAI.

El módulo VAI responde a las órdenes en los estados de WAIT o RECEP empleando el campo "info" de la variable CANStatus, durante el resto de los estados cuando se realiza transmisión CAN se envía RSTStatus. La siguiente tabla ayuda a leer las órdenes transmitidas y recibidas y los estados del módulo.

ID mensaje	ORDEN		ESTADOS VAI			
SID1 = 0x001	NTABLA	0x00	SID3 = 0x003			
	MTABLA	0x01	nFAULT	0x00	WAIT	0x05
	MTPS	0x04	RESET	0x01	RECEP	0x06
SID2 = 0x002	RSTCAN	0x02	HOLD	0x02	ERROR	0x07
	VAISW	0x03	DIR1	0x03	OFF	0x08
	CHMAPA	0x05	DIR0	0x04	CONFIG	0x09

Tabla 6.1: Tabla órdenes-SID.

Todos los datos, que son enteros, se ven en formato hexadecimal. El periodo de envío de mensajes se cambio a un segundo durante las medidas, para facilitar el trabajo.

6.2. Medición del TPS.

Para la comprobación de la sensibilidad en la adquisición de la posición del TPS, se ha conectado un LM7805 a la misma fuente de alimentación que se ha conectado el módulo VAI, simulando la batería del monoplaza de 12V. A la salida de 5V de dicho componente se tiene conectado un divisor resistivo para simular el TPS, al cual se conecta la entrada del circuito acondicionador del TPS. La medida se realizó empleando el monitor CAN de Microchip, para ver que resultados está introduciendo el nodo en el bus; para lo que se debe atender a la información vista en la columna azul oscuro de la figura 6.6 y convertirla a decimal, ya que la información en el monitor se ve en hexadecimal.

Primero se hace coincidir mediante el ajuste de la tensión de referencia del ADC la conversión de la tensión de entrada más alta, ya que como se ve en la tabla 4.6, el error debido a la ganancia

se hace mayor para esos niveles de entrada. Una vez realizado el ajuste de la tensión de referencia se conforma la siguiente tabla a partir de la variación del TPS simulado:

V_TPS_real (V)	TPS(%) acelerador	TPS(%)_id	TPS(%)_CAN	Err (%)
0,00	0	0	0	0
0,50	10	10	8	-2
1,00	20	20	18	-2
1,50	30	30	29	-1
2,00	40	40	39	-1
2,50	50	50	49	-1
3,00	60	60	59	-1
3,50	70	70	70	0
4,00	80	80	81	1
4,50	90	90	90	0
5,00	100	100	100	0

Tabla 6.2: Medidas TPS CAN.

La columna Err (%), que es el error en la medida que se ha cometido y la sensibilidad del módulo VAI, se calcula del siguiente modo:

$$Err(\%) = TPS\_CAN - TPS\_id(\%)$$

$$S_{TPS\_VAI} = \frac{\Delta\%}{\Delta Vi} = 20(\%/V)$$

Ecuaciones 6.1: Error y sensibilidad TPS CAN.

Comparando estos resultados con los medidos en la tabla 4.6, la corrección del error en la medida no es total, ya que sigue habiendo un error del 2%, pero el ajuste de la tensión de referencia ha hecho que ahora este error se dé en los porcentajes más bajos del TPS, lo cual es preferible a la forma en que estaba previamente, ya que este tipo de dispositivos, como el VAI, que da un empuje extra al coche, interesa que actúen cuando se quiere una aceleración más fuerte, evitando su activación cuando se pisa poco el acelerado al buscar aceleraciones suaves. Entonces, normalmente, se buscará la comparación del TPS en valores altos, de por lo menos el 50% por lo que este ajuste resulta suficiente para el módulo VAI teniendo un error máximo de 2% en la posición cuando el TPS es bajo, menor del 30% y un error de 1% para valores de TPS mayores del 30%. Como se aprecia en la gráfica siguiente:

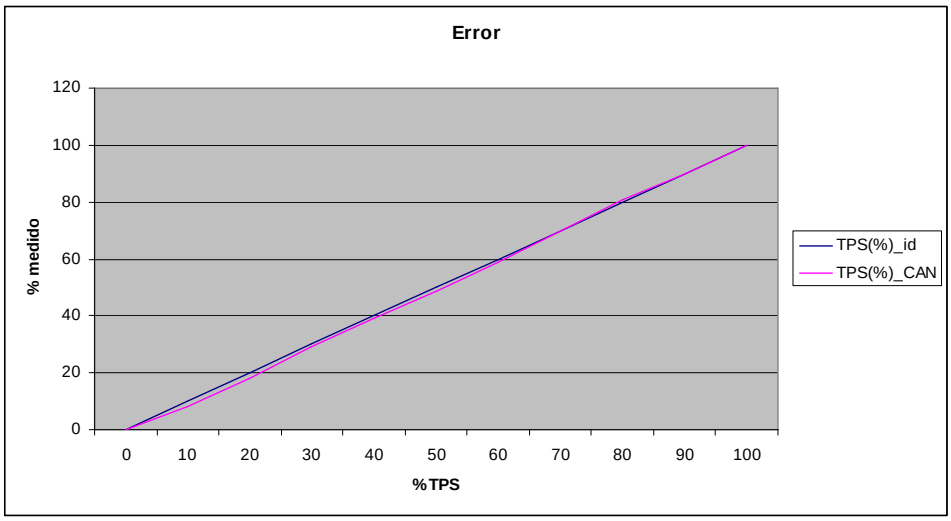


Figura 6.7: Gráfica error TPS ideal – TPS bus CAN.

6.3. Medición de las RPM.

El conjunto ATS616 + engranaje se simula mediante un generador de señal, conectándolo a los pines 2 (IC1) y 3 (Gnd) del conector de dicho sensor, como este conjunto tiene una sensibilidad de 0.5 Hz/rpm, y teniendo en cuenta el rango de revoluciones a manejar durante el movimiento del motor, el rango de frecuencias que se han de introducir con el generador van de los 4000 Hz a los 7500Hz. La medición se realiza viendo los datos del bus CAN, la columna de recuadro verde en la figura 6.6, donde D0 es la parte baja de los datos de rpm y D1 la parte alta.

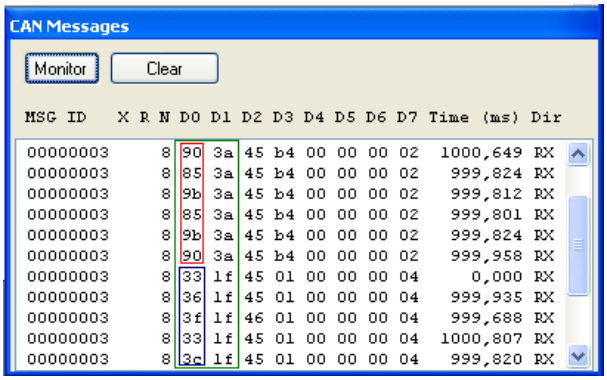


Figura 6.8: RPM en el bus CAN.

A su vez en la figura, la frecuencia máxima del ATS + engranaje está marcada con rojo y la minima con azul. Se analizan los extremos por lo visto en el apartado 5.9 para evaluar este hecho y el error máximo. Convirtiendo estas rpm a base decimal, se comprueba que el error en la medida para frecuencias altas es prácticamente igual, que para frecuencias bajas, siendo en ambos casos una variación menor de 100 rpm, que es lo deseado.

$$3A85 \rightarrow 14981(rpm) \qquad 1F33 \rightarrow 7987(rpm)$$
$$3A9B \rightarrow 15003(rpm) \qquad 1F43 \rightarrow 8003(rpm)$$
$$\Delta rpm = 15003 - 14981 = 22rpm \qquad \Delta rpm = 8003 - 7987 = 16rpm$$

Ecuaciones 6.2: Error medido de rpm.

Comprobado este error en la medida lo que se hace es cuantificarlo respecto al posicionado del paso a paso al fijar el rango mínimo de rpm para crear las tablas de mapa motor. En principio se busca el fijar este rango mínimo en 100 rpm para aprovechar la máxima resolución de posición de "runners" que da el módulo según lo calculado en el apartado 4.4.1, pero el error obtenido calculado con la ecuación 6.3 resulta ser del 11.5% para el posicionado del motor.

$$Err\_rpm(\%) = \frac{Err\_rpm}{Rango\_\min\_rpm} * 100$$

Ecuaciones 6.3: Error en % de rpm.

Por esto se fija el rango mínimo de rpm en 500 rpm resultando un error de 2.3% para la obtención de la posición de los "runners" y se dispone entonces de un máximo de 14 posiciones, que son más de las comentadas por miembros del equipo UPM-Racing. El error de la medida de rpm en porcentaje se calcula con esta misma ecuación, pero empleando el rango a medir. Dando un error en la medida de las rpm de 0.15%.

6.4. Velocidad de los "runners".

En este punto se va a analizar la velocidad del motor de posicionado de los "runners" con los medios de los que se dispone, para ello se ha empleado de nuevo el sniffer CAN de Microchip, el simulador de TPS empleado antes, que permanece constantemente por encima del 65% permitiendo el movimiento de los "runners", y que junto con el generador se simulan las rpm del monoplaza. Como el referido software indica el tiempo transcurrido entre envío y envío de tramas CAN del módulo VAI, partiendo de la posición de reposo, lo que se ha hecho es que el motor tenga que realizar su recorrido total. Viendo las tramas recibidas se sabe cuando está a 0 mm y cuando ha llegado a 180 mm, en color azul está la distancia recorrida y, por otro lado se suma el tiempo transcurrido entre tramas en color rojo. Todo esto, se aprecia en la ventana de recepción de la figura siguiente, junto con una evolución de la posición del los "runners" a lo largo del tiempo.

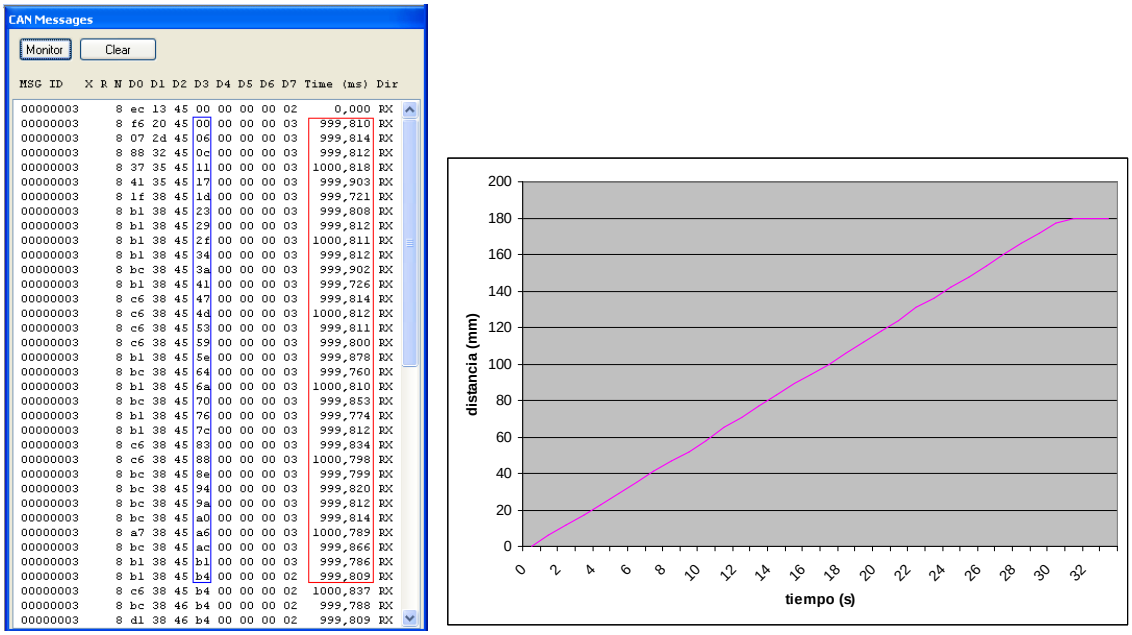


Figura 6.9: Tabla y gráfica distancia/tiempo.

De esta manera se redujo el periodo de envío para medir mejor el tiempo transcurrido, resultando que la velocidad es:

$$v = \frac{\Delta mm}{\sum t} = 5.95(mm/s)$$

Ecuaciones 6.4: Cálculo velocidad real "runners".

La velocidad obtenida de esta manera tiene un valor acorde a lo esperado debido al cálculo perfil de velocidad que se ha realizado anteriormente. No obstante, esta velocidad no tiene ninguna importancia, ya que se sabe de antemano que no se iba a conseguir la velocidad deseada de 200 mm/s, debido al material del que se disponía para la realización del proyecto.

6.5. Bus CAN.

A continuación se muestra la respuesta del módulo ante las distintas órdenes. Se ha de tener en cuenta que las órdenes RESET, VAISW y CHMAPA deben ser mandadas con SID2, y las otras tres órdenes NTABLA, MTABLA Y MTPS se han de enviar con el SID1 (tabla 6.1). Según el esquema de bus visto en el apartado 5.7.1, como se dijo en el apartado 6.1.1, el módulo VAI transmite sus tramas con SID3. Si se recibe una trama sin estar contemplado su código de orden según su filtro SID de recepción, el módulo no reaccionara ante dicha recepción.

6.5.1. Orden Reset.

Esta orden se envía desde el módulo de control para reiniciar el VAI, esto produce, que se enciendan el ADC y el IC1. Si hubo algún fallo, carga el mapa motor que estuviera en uso, carga el TPS mínimo y lleva los "runners" a la posición inicial. Este reinicio se puede seguir por la secuencia de reset que se genera empleando RSTStatus y el estado del módulo VAI.

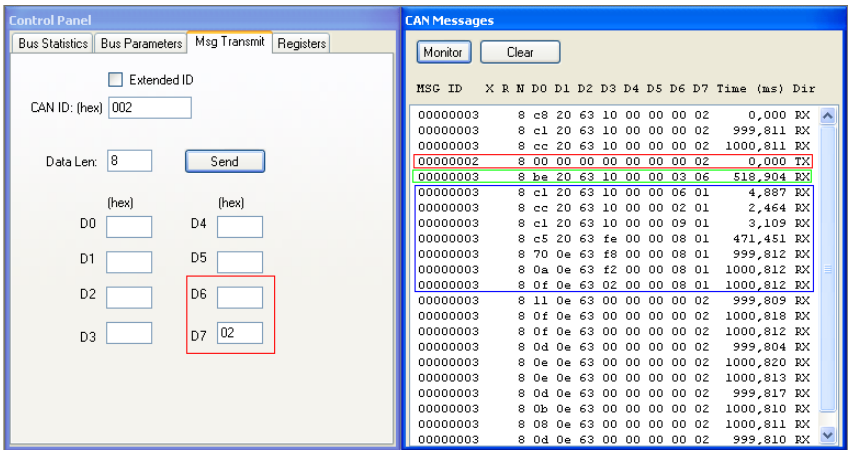


Figura 6.10: Orden RESET de CAN.

Como se aprecia en la ventana de recepción de la figura, en rojo se ve la orden mandada, en la ventana de transmisión en rojo también la orden enviada RSTCAN, los campos de datos en blanco, que no se tienen en cuenta al recibir la trama. En verde, se ve la respuesta, indicando en D6 con un 0x03 que la orden se recibió y se ejecutó. En azul se ve la secuencia de envío durante el estado de RESET hasta que llega al estado HOLD en la posición de reposo.

6.5.2. Orden On/Off.

La orden on/off (VAISW) activa/desactiva el movimiento del motor paso a paso, lleva al autómata al estado OFF, hasta que se vuelva a enviar la orden VAISW por CAN o se pulse el botón On/Off, momento en el que se recupera el estado adecuado de movimiento, DIR1, DIR0 o HOLD.

En la figura hay una secuencia de mensajes que comienza en rosa con el motor en DIR1, se envía la orden VAISW en rojo, el módulo contesta en verde en el estado de RECEP confirmando la orden, con lo que el estado del autómata se lleva a OFF y se detiene el movimiento (D3), en azul. Se vuelve a enviar la orden VAISW, el módulo contesta y recupera el movimiento (DIR1). Se envía de nuevo la orden de apagado, ahora las rpm leídas bajan

durante el estado de OFF, la distancia no varia, hasta que se envía una última vez la orden VAISW con lo que los "runners" recuperan su capacidad de movimiento esta vez en sentido contrario debido a este descenso de rpm.

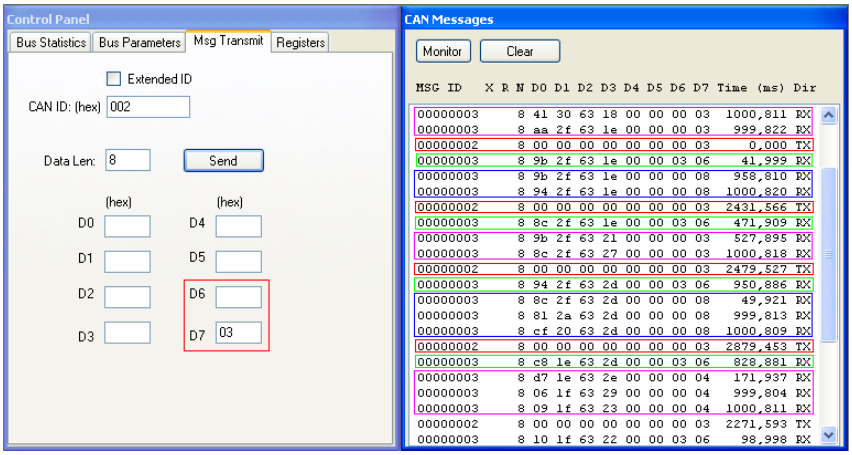


Figura 6.11: Orden On/Off de CAN.

6.5.3. Orden Cambio mapa.

Enviando esta orden CHMAPA por el bus CAN, se realiza el cambio a un mapa de motor diferente, es decir, el módulo VAI leerá de memoria FLASH una tabla de posiciones de los "runners". El nodo, notifica por CAN si la orden se ejecutó correctamente o no. Después de cambiar el mapa de motor se llevan los "runners" a la posición inicial, una vez ahí comenzarán con su funcionamiento normal. Esta secuencia se ve en la siguiente figura, en rojo la orden enviada, esta vez si el campo D6 es utilizado para enviar el número de mapa al que se cambia, en verde la contestación, en azul la secuencia de reposicionado de los "runners" y vuelve a recuperar el funcionamiento. En rosa se puede ver la información del número de mapa en uso.

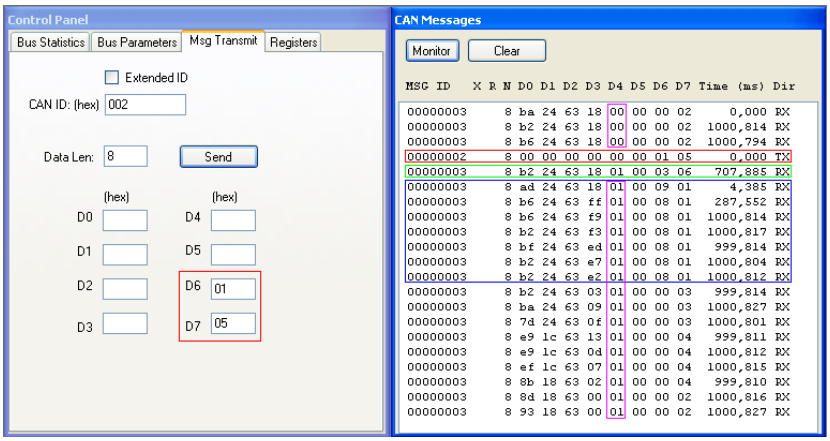


Figura 6.12: Orden CHMAPA de CAN.

Si se envía, como en la secuencia siguiente un número de mapa no admitido (rojo) o el número de mapa enviado es el que ya esta en uso, el módulo igualmente contesta indicando que no se hizo un cambio de mapa (0x00) en D6 (verde) y los “runners” continúan con el movimiento que tuvieran. En rosa se ve que el número de mapa actual no cambia, aunque en D6 de la orden se solicita el cambio al mapa 7, que no existe, pues los mapas son del 0 al 5, seis en total.

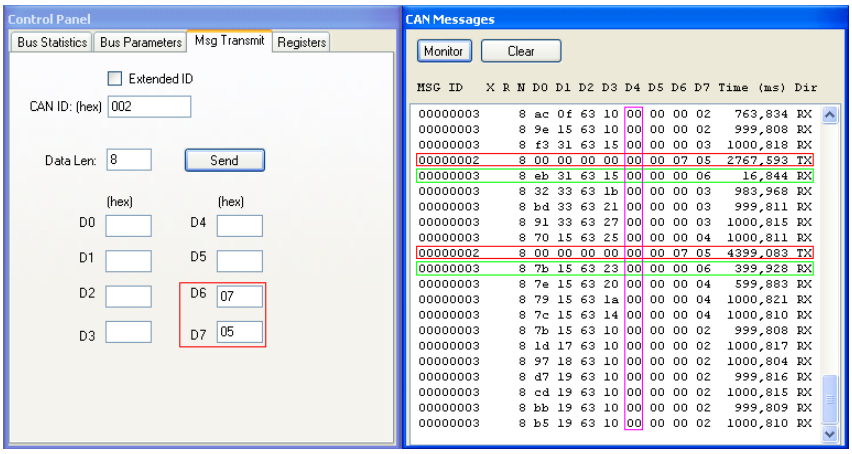


Figura 6.13: Orden errónea CHMAPA.

6.5.4. Orden Tabla nueva.

Con el envío de esta orden NTABLA, se crea una nueva tabla de posiciones para los “runners”. Cuando se envía por primera vez, para la creación de esta nueva tabla solo se atienden el número de mapa motor a crear y el número máximo de posiciones, si ambos están dentro de rango el módulo pasa a estado WAIT y contesta con un 0x01 en D6 para indicar que la tabla está en creación. Después de esto permanece a la espera del resto de tramas para conformar la tabla, que se deben ir mandando también con la orden NTABLA sin importar que el máximo de posiciones o el número de mapa motor cambien; la tabla se formará con los datos enviados en el primer mensaje. Además, en este inicio de creación de tabla no se miran las rpm y mm enviados la primera posición es siempre 0 mm mientras no se pase de 8000 rpm.

La segunda posición debe aumentar un mínimo de 1 mm y 500 rpm respecto a la posición primera, las siguientes posiciones se deben seguir completando según esta norma hasta alcanzar el número máximo de posiciones indicado en la primera trama enviada con esta orden, siendo el número mínimo de posiciones 2 y el máximo de posiciones es 14. Si eso no se cumple, el módulo contesta con un 0x02 para señalar este error, si se envían tres posiciones erróneas seguidas, el módulo contesta con 0x00, realiza un reset de posición del motor y carga la tabla

anterior en uso. Si la secuencia de la tabla es correcta, al final el módulo contesta confirmando la creación de una nueva tabla (0x03). Si la tabla creada es un número de mapa motor distinto al que está en uso, el mapa motor cambia al del mapa motor creado y finalmente se realiza el reset de la posición de los “runners”.

En la figura 6.14 se ve la secuencia de creación de una tabla nueva, en ella se envían posiciones incorrectas para ver la respuesta del módulo cuado esto sucede. En rojo están las órdenes de envío de las distintas posiciones. Después del primer envío de orden NTABLA el módulo deja de enviar sus datos periódicamente; en la secuencia se envían hasta tres mensajes erróneos, intercalados con posiciones correctas de tal modo que la nueva tabla finalmente se crea correctamente, y se guarda en la memoria FLASH, realizando, en azul, el reposicionamiento de los “runners” a su situación inicial. Se ve, que el mapa de motor cambia al del mapa motor creado; en este caso, se cambia al mapa motor 1 y el mapa motor 0 permanece en memoria igual que estaba. En la tabla siguiente se muestra cuales son las posiciones de los “runners” y los valores umbral rpm de cambio de posición de una tabla de posiciones de un mapa motor donde se ven las rpm, mm, número de tabla y posiciones máximas en sus valores decimales y hexadecimales.

RPM		mm		MAXPOSC		n_mapa	
dec	hex	dec	hex	dec	hex	dec	hex
8000	0x1F40	0	0x00	6	0x06	1	0x01
8500	0x2134	16	0x10	6	0x06	1	0x01
9000	0x2328	20	0x14	6	0x06	1	0x01
11000	0x2AF8	24	0x18	6	0x06	1	0x01
12250	0x2FDA	66	0x42	6	0x06	1	0x01
15000	0x3A98	180	0xB4	6	0x06	1	0x01

Tabla 6.3: Detalle de una tabla de 6 posiciones.

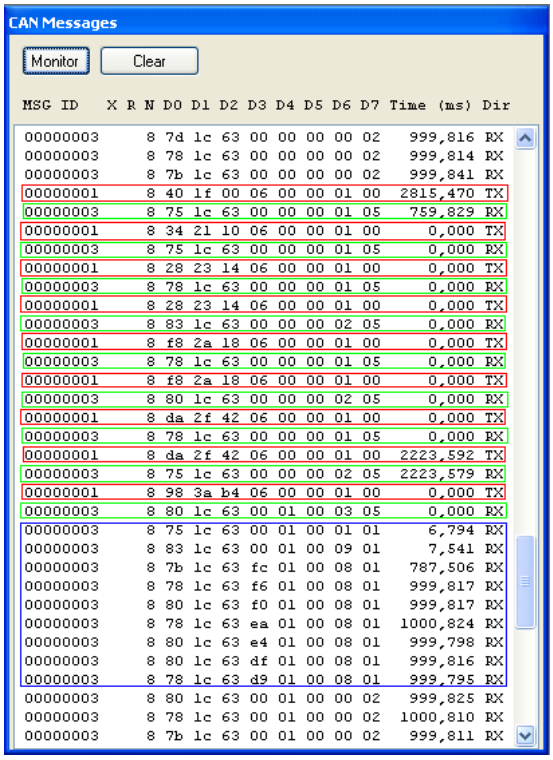


Figura 6.14: Secuencia de creación de un mapa motor.

En la figura siguiente se muestra una secuencia de creación de tabla errónea, tras enviar dos posiciones correctas, se envían tres posiciones erróneas seguidas. El módulo contesta con el código de posición incorrecta en D6 (0x02) dos veces, y en la tercera envía un 0x00 y realiza el reset de posición de los “runners”, cargando la tabla anterior en uso. En este caso D4 vale 0x00 que señala que se estaba en el número de mapa motor 0 al iniciar y no ha cambiado cuando el número de mapa motor que se estaba intentando crear era el 1.

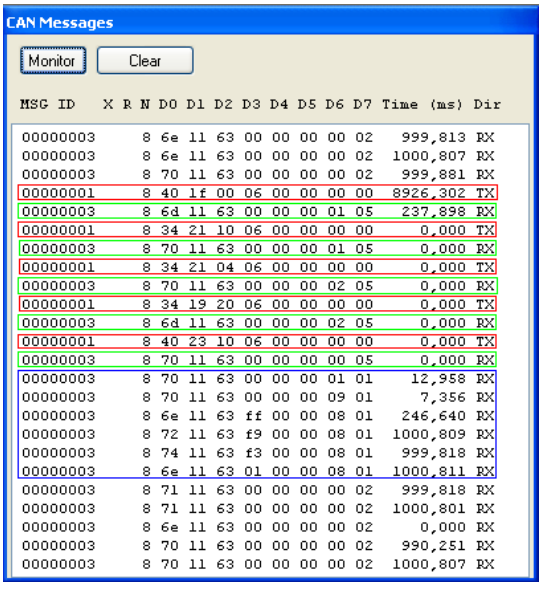


Figura 6.15: Secuencia errónea de creación de un mapa motor.

6.5.5. Orden Modifica tabla.

En caso de error al introducir una tabla de posiciones nueva o cuando se desee hacer algún ajuste sobre alguna posición, para evitar tener que introducir la tabla entera de nuevo, esta orden MTABLA permite modificar una posición de la tabla en uso. En los datos se ha de indicar la orden, rpm, mm y la posición a modificar. La nueva distancia de los "runners" debe ser 1 mm superior que la posición anterior y 1 mm menor que la siguiente posición de la tabla. Las rpm nuevas deben ser 500 rpm mayor que la posición anterior y 500 rpm menor que la siguiente posición de la tabla. Tanto en el caso de que el cambio se realice correctamente como en el caso de no modificarse la posición de la tabla, el hecho se notifica por CAN. En la figura siguiente se ve la orden MTABLA en rojo y la respuesta en verde, tras esto el motor se lleva a la posición de inicio con el reset del motor paso a paso, en azul.

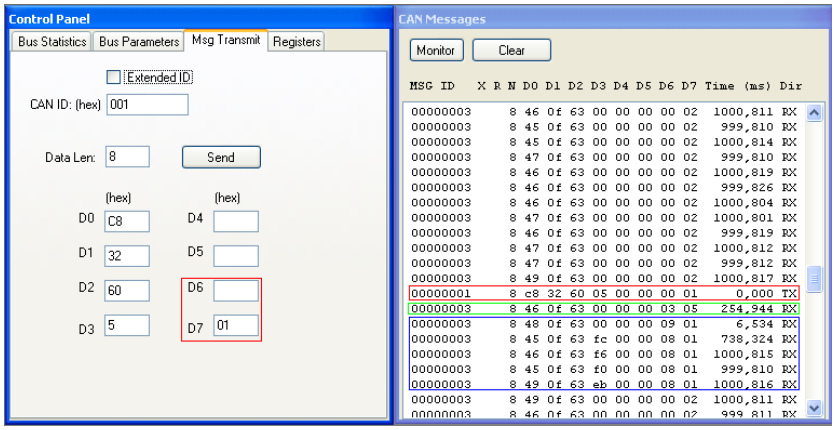


Figura 6.16: Orden MTABLA de CAN.

En la figura de arriba, se ha modificado la posición quinta de la tabla de posiciones tabla 6.3. En la figura de abajo sobre esta misma tabla se introduce una posición errónea lo que lleva al módulo a realizar la respuesta acorde a esta no modificación de la tabla. Finalmente los "runners" son igualmente reiniciados. En rojo se ve el envío de la orden, en verde la contestación del módulo VAI y en azul el reset. En rosa, se ve el número de posición a modificar fuera del rango de posiciones de la tabla en uso.

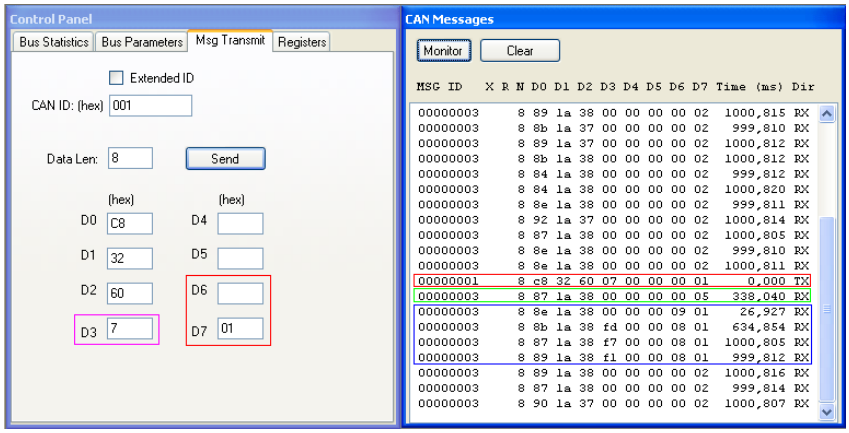


Figura 6.17: Orden errónea MTABLA.

6.5.6. Orden Modifica TPS mínimo.

Esta orden, MTPS, modifica el valor umbral de TPS a partir del cual los "runners" podrán reducir su longitud acorde a las rpm del monoplaza. Cuando se recibe esta orden, si el nuevo valor de TPS esta entre 1 y 100, se guarda el nuevo TPS mínimo en memoria no volátil y se realiza el reset de la información leída de memoria FLASH y de la posición del motor.

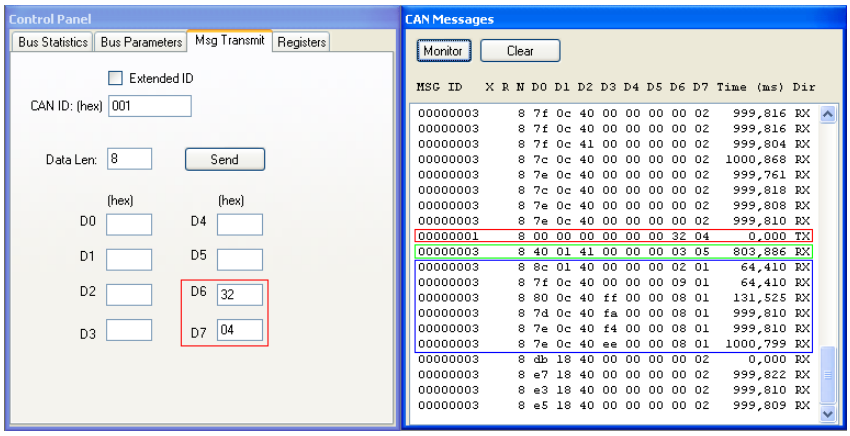


Figura 6.18: Orden MTPS de CAN.

En la figura anterior se ha modificado el valor del TPS mínimo al 50%, en hexadecimal 0x32, lo que se refleja en el campo D6 de la orden enviada. En la siguiente figura en el campo D6 se envía 0xFF, que esta fuera del rango de los valores admisibles para el TPS, por lo que el autómata no registra este nuevo valor de TPS. Para los dos casos, la interpretación de los colores es la misma, rojo es la orden enviada, verde es la respuesta del nodo VAI y en azul la secuencia de reset.

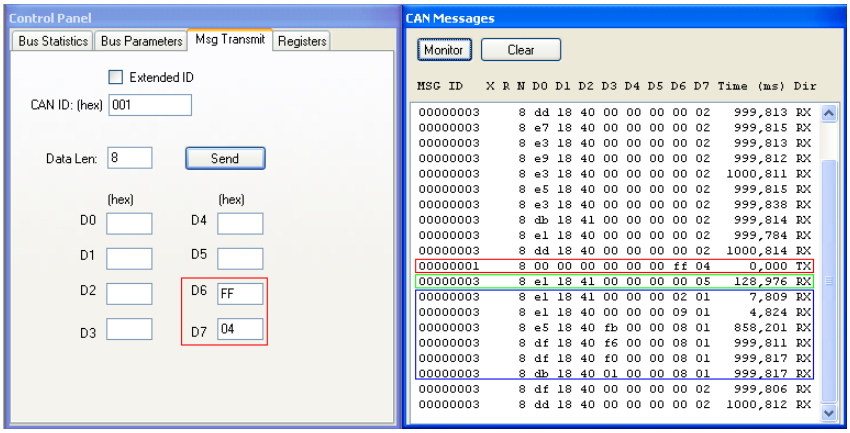


Figura 6.19: Orden errónea MTPS.

6.6. Posicionado de los “runners”.

La posición de los “runners” se obtiene con la misma situación que para obtener su velocidad, se ha cargado la tabla de posiciones tabla6.3 en memoria con número de mapa 0. El TPS se pone por encima del 65% para que el motor pueda variar su distancia y se van incrementando las rpm para ver las distancias obtenidas empleando el MCP2515 CAN.

Fijándose en la distancia obtenida por las tramas CAN, que es la que hay entre la posición de reposo de los “runners” y la posición de los mismos en función de las rpm del monoplaza. Se compone la siguiente gráfica de desplazamiento de los “runners” en función de las rpm del motor:

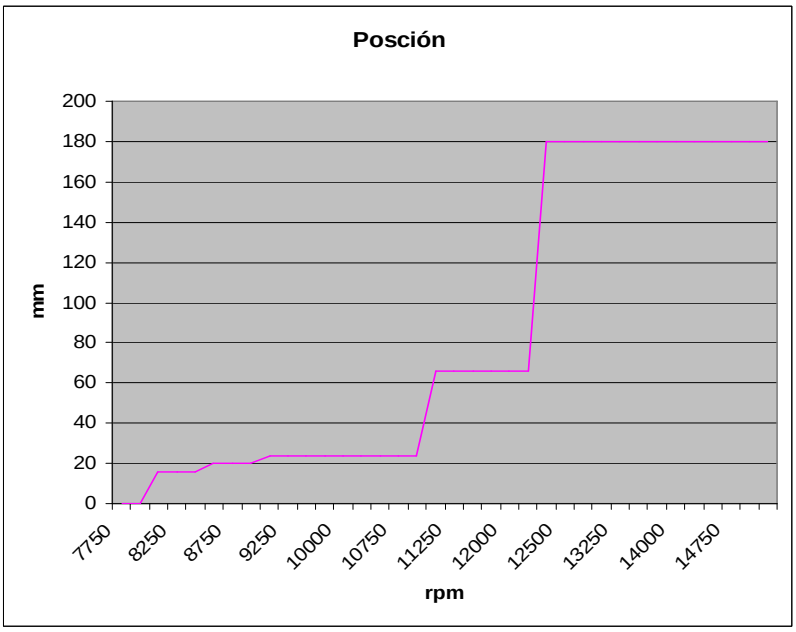


Figura 6.20: Distancia motor paso a paso-rpm.

La variación de la longitud de los “runners” se realiza de dos formas distintas: Primero, mientras el TPS esta por encima de umbral de TPS mínimo el sistema mueve los “runners” hacia cualquier dirección acorde a lo dispuesto en la tabla de posiciones del mapa de motor y según las rpm del monoplaza. Sin embargo, segundo, si el TPS no está por encima del umbral mínimo la longitud de los “runners” solo puede aumentar. Esto resulta importante en el comportamiento del vehículo ya que, por ejemplo, cuando se quiera emplear el freno motor, situación en la que las rpm son elevadas pero no se actúa sobre el acelerador, no interesa que el sistema VAI entregue más potencia al motor acelerándolo cuando se quiere frenar. Si las rpm bajan, la longitud de los “runners” si aumentará así los “runners” estarán en la posición adecuada para cuando se vuelva a acelerar.

En consecuencia, según la información de la tabla y la programación del software, en cada posición se indica la distancia del motor paso a paso respecto a la posición de reposo, en la que permanecerá hasta que se superen sus rpm, momento en el cual se incrementa una posición, se lleva al motor paso a paso a la distancia de la nueva posición actual de la tabla y permanece en ella hasta que las rpm del monoplaza vuelvan a superar el nuevo umbral o bajen del umbral de la posición anterior, en cuyo caso se baja una posición variando de nuevo la longitud de los “runners”.

6.7. Futuras líneas de desarrollo.

Un trabajo que se debe realizar es la instalación del PCB VAI_4 y el motor paso a paso en el monoplaza para poder someter al mismo a un banco de pruebas y ver los resultados de la variación de la geometría del motor al cambiar la longitud de los “runners”. Pudiendo situar la electrónica alejada del motor paso a paso estando este fijo en un punto y mediante el sinfín estire y recoja los “runners”

Las futuras líneas de desarrollo sobre este proyecto en concreto están partidas en dos bloques, el primero consiste en lo referente al bus CAN, ya que para la comprobación de funcionamiento y diseño del módulo VAI se simularon más módulos conectados al bus, por lo que si estos nodos no están, estas propiedades del módulo VAI no funcionaran. Por lo que habrá que integrar las órdenes el los otros módulos del bus y la forma en que se deben transmitir los datos al módulo VAI o viceversa, adaptar el modo en el que el módulo VAI recibe y envía los datos del bus.

Además se deben configurar filtros, mascaras y el identificador de mensaje enviado del VAI acorde a los identificadores ya asignados al resto de nodos del bus. La velocidad del bus

también es susceptible a ser modificada, ya que en el monoplaza puede que tengan el bit-timing distinto al configurado en este nodo.

El otro bloque de desarrollo se refiere al modelo de motor paso a paso empleado, ya que del que se dispone no cumple con la especificación de velocidad y, al cambiar de modelo de motor es posible que sea necesario modificar más hardware como el DRV8805, que es únicamente para motores unipolares. También seria necesario, seguramente, alguna fuente de alimentación y una etapa de potencia para poder suministrar la corriente necesaria a las bobinas del nuevo paso a paso, ya que después realizar búsquedas de motores paso a paso para el proyecto, los motores que presentan unas características apropiadas para el proyecto necesitan un mínimo de 24V y 1 A por fase.

El cambio de motor paso a paso también tendría una repercusión en el software, habiendo que reconfigurar el bloque software Control Motor, con los cambios que eso conlleva, periodo de comparación de timer4 para controlar la distancia recorrida, OC5 para controlar la velocidad y el timer 2 ya que el perfil de velocidad también cambiará.

6.7.1. Motor Nanotec.

En esta sección se presenta un motor apropiado para el desarrollo del sistema a la velocidad adecuada. Después de búsquedas entre distintos fabricantes de motores paso a paso lineales, se ha optado por Nanotec, que posee una amplia gama de modelos, de entre estas gamas, la L41 se adapta bien al proyecto por su reducido peso y tamaño, además dentro de esta gama los modelos del tipo T5x5 son capaces de desarrollar velocidades superiores a las especificaciones dadas en el apartado 1.8 por el INSIA. Por otra parte, aunque en el catálogo actual los modelos son motores bipolares, en versiones anteriores de esta gama si hay motores unipolares y en su web se invita al cliente a preguntarles por la posibilidad de realización de motores para aplicaciones específicas. Por lo que sería ideal la elección del modelo unipolar antes que el bipolar, ya que de esta manera no seria necesario reemplazar el driver DRV8805. Además este fabricante te ofrece la opción de comprar el motor con un ENCODER que instalan opcionalmente.

Para la elección del modelo concreto del actuador lineal se va ha mirar su gráfica de pull-in rate de este tipo de motores (T5x5) dentro de la gama L41. Se busca que la empuje del motor sea mayor que el conjunto motor + “runners”. Con esto se busca dar mayores posibilidades de integración del sistema en el monoplaza.

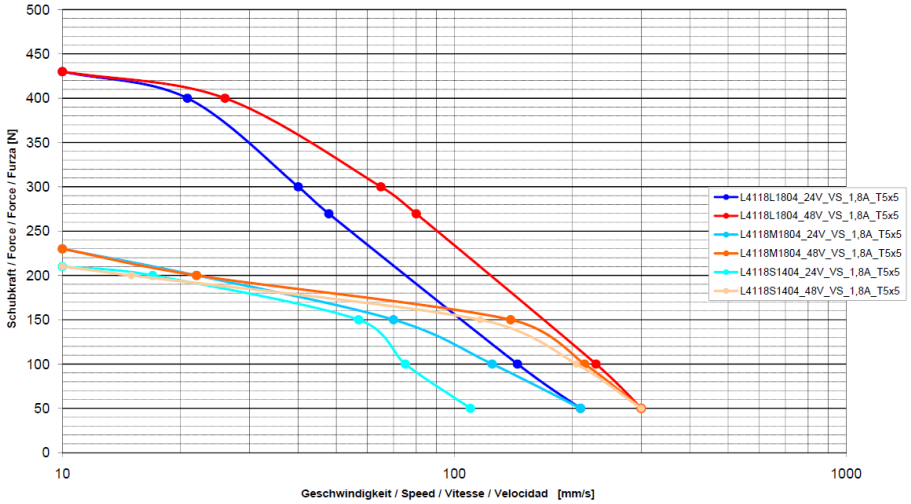


Figura 6.21: Gráfica fuerza-velocidad Nanotec.

A la vista de la gráfica basta con elegir un modelo que tenga la velocidad suficiente, ya que el peso de la carga ("runners") es 0.4 kg y el peso de estos motores es como máximo 0.35 kg, lo que haría necesaria una fuerza en el motor mayor a unos 10N. Por ello se analiza si el modelo apropiado seria el L4118L1804 T5x5 cuando se conecta a una fuente de tensión de 24 V.

Las características más importantes de este motor son:

característica	L4118L1804 T5x5
Tensión (V)	24-48
Empuje max. (N)	250
Fuerza de sujeción (N)	40
resolución de paso (mm/paso)	0.025
Pull-in rate max. (mm/s)	250
Peso (kg)	0.35

Tabla 6.4: Características motor Nanotec L4118L1804.

Por lo que en cuanto a la inercia de las masas el peor de los casos sería que el motor estuviese desplazándose junto con los "runners" sumandose ambas masas, con lo que según la ecuación 4.1 resulta una fuerza de inercia total de 7.35 N. Como lo que se va a buscar es que el desplazamiento en un milímetro se realice a una velocidad de 200mm/s, se divide el espacio recorrido ente la velocidad y se sabe que se dispone de un tiempo de desplazamiento igual a 5 ms.

Lo siguiente es obtener el tiempo de aceleración hasta dicha posición, empleando las ecuaciones del cálculo de perfil de velocidad trapezoidal vistas en el capitulo 3 (ecuaciones 2.6). La gráfica pull-in que facilita el fabricante está definida en fuerza-velocidad. En este caso las ecuaciones son las mismas, el cambio es la variable que en vez de frecuencia de paso se emplea la

velocidad, así T(v) que se idealiza en dos partes y TL(v) es constante y puede ser el motor + "runners" o solo estos últimos. Los tiempos de aceleración más cortos obtenidos para los dos posibles casos de carga son:

$$t(v \le 20) = \int_0^{20} \frac{1}{T_1(v) - T_L(v)} \partial v$$

$$t(20 < v \le 220) = \int_{20}^{220} \frac{1}{T_2(v) - T_L(v)} \partial v$$

$$T_1(v) = 400(N) \Rightarrow v \le 20(mm/s)$$

$$T_2(v) = 435 - 1.75(N) \Rightarrow 20 < v \le 220(mm/s)$$

$$T_{L\min}(v) = 3.92(N) \rightarrow t_{\min} = 12.7ms$$

$$T_{L\max}(v) = 7.35(N) \rightarrow t_{\max} = 24.7ms$$

Ecuaciones 6.5: Tiempo aceleración y elección de carga Nanotec.

Con los tiempos mínimos obtenidos para cada una de las situaciones de carga se optaría por dejar el motor fijo y que sean los "runners" los que se muevan por la acción del sinfín. Ahora bien, aún seleccionando la disposición de carga mínima, se ve que la resolución que se buscaba no se puede alcanzar, la solución que se realiza es fijar esta nuevamente en 10 mm para la realización del perfil y se emplea el tiempo mínimo de los dos obtenidos como referencia de tiempo de aceleración y de parada. De esta manera se dispone de un tiempo total de 50 ms para recorrer 10 mm a una velocidad media de 200 mm/s, para una vez obtenido el tiempo real de aceleración se debe establecer el número de pasos a velocidad máxima. Y así finalmente obtener la velocidad media buscada.

Se fijan los tramos de aceleración, en este caso se van ha realizar 5 tramos. E igual que para obtener los tiempos mínimos anteriormente se obtienen los tiempos para cada tramo de aceleración.

$$v_1 = 50 - 0(mm/s) \rightarrow t_{acc1} = 1.31(ms)$$

$$v_2 = 150 - 50(mm/s) \rightarrow t_{acc2} = 4.06(ms)$$

$$v_3 = 190 - 150(mm/s) \rightarrow t_{acc3} = 3.06(ms)$$

$$v_4 = 210 - 190(mm/s) \rightarrow t_{acc4} = 2.50(ms)$$

$$v_5 = 220 - 210(mm/s) \rightarrow t_{acc5} = 1.84(ms)$$

$$t_{acc\_total} = t_{acc1} + t_{acc2} + t_{acc3} + t_{acc4} + t_{acc5} = 12.77(ms)$$

Ecuaciones 6.6: Cálculo de tiempo de aceleración de Nanotec.

Para cada valor de velocidad el motor tiene una potencia determinada, con esta distribución, durante el primer periodo de aceleración que se alcanzan los 50mm/s el motor dispone de un

empuje de 260N, en el siguiente tramo su empuje es de 100N, luego sucesivamente 70N, 60N y finalmente 50N.Esta velocidad se convierte en frecuencia de excitación de paso dividiendo la velocidad entre la resolución del paso del motor y se calcula el número de pasos a partir del cual se ha de cambiar la frecuencia de pasos. Este número se ha de redondear en pasos completos.

$$f_{pasos} = \frac{v_i}{mm/paso}(paso/s)$$
$$N\_pasos\_acc = t_{acc} * f_{pasos}$$
$$N\_f_1 = 2.62\,pasos$$
$$N\_f_2 = 24.36\,pasos$$
$$N\_f_3 = 23.256\,pasos$$
$$N\_f_4 = 21\,pasos$$
$$N\_f_5 = 16.192\,pasos$$
$$N\_pasos\_acc = 3+24+23+21+16 = 87\,pasos$$

Ecuaciones 6.7: Cálculo del número de pasos por tramo Nanotec.

Con las condiciones de la velocidad media que se establecen, empleando un perfil trapezoidal con los dos tiempos iguales se calcula empleando las ecuaciones 2.2, y fijando un desplazamiento de 10mm, se obtiene un tiempo de velocidad constante y finalmente la velocidad media de la carga con las ecuaciones 2.5.

$$N\_total = \frac{r = 10(mm)}{mm/paso} = 400\,pasos$$
$$N\_v_{cte} = 400 - 2*87 = 226\,pasos$$
$$t\_v_{cte} = 226\frac{1}{f_5} = 25.68(ms)$$
$$t\_v_1 = 3\frac{1}{f_1} = 3.75(ms)$$
$$t\_v_2 = 24\frac{1}{f_2} = 4(ms)$$
$$t\_v_3 = 23\frac{1}{f_3} = 3.02(ms)$$
$$t\_v_4 = 21\frac{1}{f_4} = 2.5(ms)$$
$$t\_v_5 = 16\frac{1}{f_5} = 1.81(ms)$$
$$v_{media} = \frac{400*0.025}{2*15.08*10^{-3} + 25.68*10^{-3}} = 179.08(mm/s)$$

Ecuaciones 6.8: Cálculo de la velocidad media Nanotec.

Viendo la velocidad media obtenida y sabiendo la velocidad máxima del motor Nanotec, se mejorará la velocidad media elevando la velocidad máxima del motor; pero como ya se está empleando, se hace necesario alimentar el motor con 48 V, efecto que produce un mayor empuje y una mayor velocidad máxima repitiéndose todo el proceso de diseño del perfil con una nueva obtención de la función T(v). Y como carga de inercia se empleará también únicamente la masa de los "runners". Otra opción para mejorar la velocidad media sería incrementar de nuevo la distancia fijada de 10 mm, para saber cuantos son los pasos que habrá que dar para obtener la velocidad deseada, a partir de la velocidad media se suman posiciones y tiempos para ir incrementando la velocidad media. En la representación de la función se aprecia el incremento de la velocidad media del perfil al incrementar la longitud del desplazamiento.

$$v_{media} = \frac{(400 + x) * 0.025}{2 * 15.08 * 10^{-3} + 25.68 * 10^{-3} + x * 1.1 * 10^{-4}}$$

Ecuaciones 6.9: Obtención de pasos a v_cte Nanotec.

Para valores de x de 0 a 600 se obtiene la gráfica siguiente, en el eje y esta la velocidad en mm/s y el eje x expresa el número de pasos a sumar al desplazamiento:

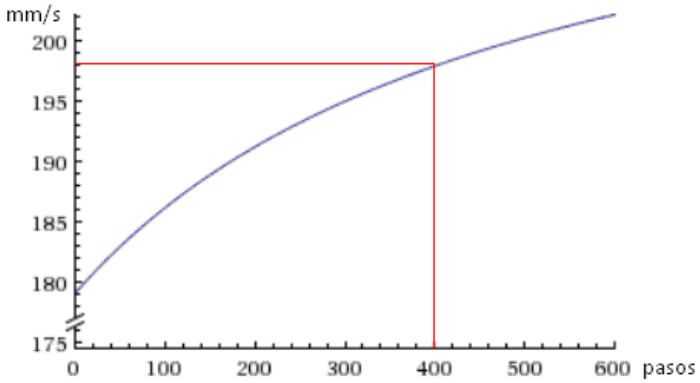


Figura 6.22: Gráfica tendencia de velocidad-nº pasos.

Si se hace que el desplazamiento se incremente en 400 pasos más a velocidad máxima, se obtiene una velocidad media de 197.44 mm/s. Realizándose desplazamientos de 20 mm en 100(ms) una velocidad media de recorrido de aceptable a lo deseado de 200mm/s, pudiéndose mejorar incrementando de nuevo el número de pasos.

El ajuste de esta resolución de 20 mm para mantener lo mayor posible la velocidad media constante y permitir que los "runners" se detengan en cada milímetro sería trabajo software. Empleando la distancia recorrida e iniciando en el momento adecuado una secuencia de parado

del motor paso a paso, adecuándola al punto del perfil en el que se encuentre. Teniendo en cuenta que desde la velocidad máxima se requieren 87 pasos para detener correctamente el motor paso a paso.

6.7.2. Cambios en PCB.

Si no se pudiese encontrar un modelo de motor Nanotec unipolar. Con el uso de un motor bipolar como el seleccionado habría que cambiar el DRV8805 por el DRV8412 por ejemplo, el cual permitiría realizar el control del motor conectándolo a la fuente de tensión adecuada.

Además si se tuviese un ENCODER se conectaría la señal I de este a la entrada del timer 4.

6.7.3. Cambios software.

En el software habría que retocar la tabla del perfil de velocidad e introducir líneas de código nuevas para el control del perfil de velocidad dentro del timer 4, y para adaptar la parada del motor forzando la secuencia de frenado del perfil así como algunos registros de configuración del mismo. La función ControlMotor(estado) facilita el cambio de driver, ya que solo habría que rehacer esta función acorde a los estímulos que necesite el nuevo driver.

Este control software haría que para desplazamientos mayores o iguales a 20mm la velocidad del movimiento fuese de 197 mm/s, es decir, el límite superior se respetaría. Y en caso de ser desplazamientos menores de 20mm la velocidad media seria menor, debido a esta “perdida de resolución”.

Presupuesto

Coste de componentes**:**

Item	Cantidad	Ref.	Valor	Precio
1	17	C1,C2,C3,C4,C5,C6,C8,C10,C11,C12,C13,C14 ,C15,C16,C20,C21,C27	0.1uF	1,2
2	4	C7,C18,C19,C28	1uF	0,56
3	3	C9,C24,C25	10uF	0,33
4	1	C17	100uF	0,05
5	1	C22	330uF	0,09
6	1	C23	15uF	0,05
7	1	C26	0.33uF	0,11
8	6	D1,D9,D10,D11,D12,D13	LED	1,8
9	5	D2,D3,D7,D8,D15	1N4148	0,05
10	3	D4,D5,D6	6V8_zener	0,0292
11	1	D14	1N5817	0,09
12	1	D16	4V2_zener	0,03
13	1	J1	CON_debug	2,53
14	1	J2	SFC_CON	0,67
15	1	J3	TPH_CON	0
16	1	J4	MOTOR_CON	1,75
17	1	J5	ATS616_CON	1,2
18	1	J6	JUMPER	0
19	1	J7	JACK VCC12	0,2
20	1	J8	JACK VM	0,2
21	1	P1	CON_CAN_DB9	0,55
22	1	R1	0	0
23	6	R2,R19,R20,R21,R22,R23	470	0,18
24	3	R3,R6,R7	50	0,24
25	4	R4,R9,R11,R27	10k	0,13
26	4	R5,R17,R18,R25	200	0,6
27	1	R8	120	0,04
28	2	R10,R12	4K	0,45
29	2	R13,R15	5k	0,08
30	2	R14,R16	3.3K	0,08
31	2	R26,R24	1k	0,18
32	1	SW1	RST	0,2
33	1	SW2	SW ON/OFF	0,2
34	1	SW3	interruptor VM	0,9
35	1	SW4	interruptor Vcc12	0,9
36	1	TP1	TPHacond	0
37	1	TP2	Vref+	0
38	1	U1	PIC32MX795F512	13,74
	1	U2	MCP2551	0,5+1,18
39	1	U3	LM358	0,5+0,79
40	1	U4	DRV8805	4,17
41	1	U5	OP-07	0,5+1,04
42	1	U6	LM7805C/TO220	0,15
43	1	U7	LM3940/TO220	1,15
44	1	U8	NMH0512S	13,74
45	1	PCB	Doble Cara	10,16
SUBTOTAL COMPONENTES €				58,7792

Coste de material:

Item	Ref.	Precio
1	PICkit3	45,48
2	CAN bus MONITOR MCP2515	46,71
3	Portescap 56DBM10B2U-L	150
SUBTOTAL MATERIAL €		242,19

Coste de Ingeniería:

Coste de hora de ingeniería: 10€

Horas de trabajo distribuidas:

- Estudio inicial: 150 horas.
- Desarrollo hardware: 300 horas.
- Desarrollo software: 300 horas.
- Depuración y pruebas: 400 horas.

Horas totales	Precio
1150	10
SUBTOTAL INGENIERÍA €	11500

Coste total:

SUBTOTAL COMPONENTES €	58,78
SUBTOTAL MATERIAL €	242,19
SUBTOTAL INGENIERÍA €	8625
TOTAL €	11800,97

Conclusiones.

Divido la redacción de Conclusiones en dos partes. La primera de ellas corresponde a las observaciones o dificultades con que me he encontrado al trabajar con gusto en este proyecto de "fin de carrera" en el que me he embarcado; mientras que la otra parte viene a ser la constatación de las cualidades que he encontrado en el proyecto en su conjunto.

Por tanto, redacto estas conclusiones en primera persona, ya que se trata, más que de verdaderas conclusiones, de opiniones o apreciaciones, una vez concluido el trabajo.

I.- Observaciones o dificultades encontradas.

En primer lugar he de consignar el esfuerzo que me ha supuesto el choque con mi propia incorporación al trabajo de un proyecto realmente existente. De hecho, me encontré con unas especificaciones que, en principio, no me parecieron muy concretas, por lo que tuve que imaginarme la posible situación de un bus CAN en el conjunto del proyecto.

En segundo término, he sentido la ausencia de contacto directo con el monoplaza en el que tendría que encajar el sistema en el que se ubicaría el resultado de mi trabajo, comprobando cómo es el TPS existente y, puesto que a mi entender, ya debe existir en la realidad del coche alguna forma de medir las rpm del motor como es el sistema con que cuenta el monoplaza.

Por otra parte, tampoco pude contar con la suficiente o apropiada información, por parte del INSIA, sobre el movimiento de los "runners" del monoplaza.

Finalmente, lamento que no haya sido posible adquirir un motor con la velocidad adecuada junto con el tornillo sinfín y un ENCODER y tener que usar el motor Portescap como único motor lineal disponible, lo que ha limitado la velocidad del movimiento alcanzable a la hora de poder comprobar las cualidades del proyecto.

II.- Cualidades alcanzadas.

Considero una buena opción trabajar la variación de la geometría del motor, puesto que es desde este módulo de donde se puede obtener mejoras para la competición, tales como cambios en el rendimiento y en la potencia, al conseguir mejoras en las curvas de par y potencia para el caso del motor de combustión.

Igualmente, llevar a efecto el protocolo de comunicaciones CAN me parece un elemento importante en un vehículo diseñado para la competición, ya que permite el control de muchos de sus elementos con solo una conexión sencilla y robusta.

En mi opinión, antes que nada se debería desarrollar un proyecto que organice el bus del monoplaza con los nodos que ya existan o los que se deseen integrar asignando, por lo menos, los distintos números identificadores de mensaje, incluso forzando la línea de desarrollo del proyecto acorde a sus necesidades de operación.

Por tanto, yo estoy convencido de que, mientras no se reemplace este motor Portescap por otro motor de mayor velocidad, al desplazarse muy despacio este módulo VAI sigue siendo una buena opción para el ajuste de mapas de motor en un banco de pruebas. Siendo su acción muy limitada para la propia competición.

Referencias Bibliográficas.

Documentación variada descargada o consultada directamente en las webs, como datasheets, notas de aplicación, catálogos, etc.

http://www.upmracing.es/

http://www.insia-upm.es/

http://students.sae.org/

http://www.fsae.com/

http://www.youtube.com/

http://www.yamaha-motor.eu/

http://www.honda.es/

http://www.microchip.com/

http://www.ti.com/

http://www.analog.com/

http://es.rs-online.com/

http://es.farnell.com/

http://www.portescap.com/

http://en.nanotec.com/

http://www.exlar.com/

http://www.bosch-semiconductors.de/

http://lernsystem.original-marken-partner.de

http://es.wikipedia.org/

http://www.h2wtech.com/

http://www.trinamic.com/

http://www.servomech.com/

http://www.mecmod.com/

http://www.progressiveautomations.com/

- Manual CAN2.0 Bosch.
- Manual Orcad.
- Manual Layout.
- Manual compilador C32.
- Manual MPLABX.
- Manual Microchip CAN Monitor MCP2515.
- Documentación aportada por el INSIA.

7. ANEXO I.

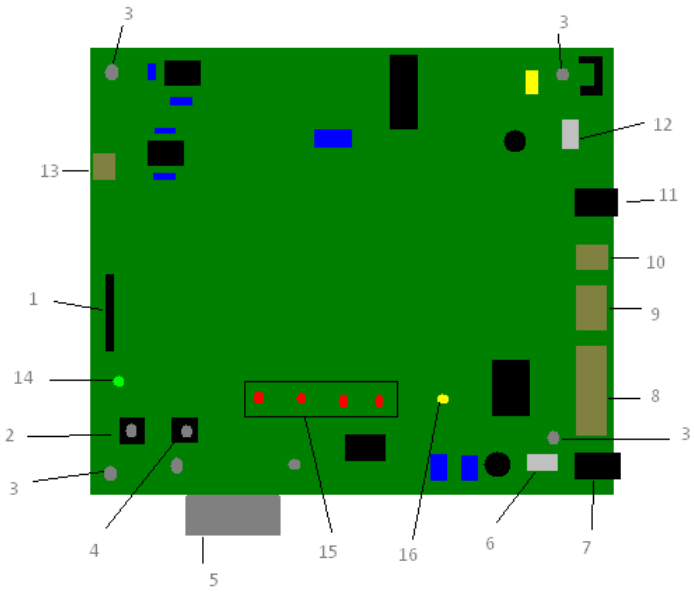
MANUAL DE USUARIO

1. Conexión y montaje.

La PCB VAI se puede ubicar donde se desee en el monoplaza adecuando la longitud de los cables conectores de los sensores, bus CAN y del motor paso a paso. El motor debe permanecer fijo, siendo tarea del equipo de ingenieros del vehículo decidir la configuración del motor paso a paso y su varilla sin fin. Lo que se debe considerar al realizarse esta configuración es:

- En DIR0, el paso a paso mueve los "runners" buscando que la distancia entre su situación actual y su posición de reposo sea 0 mm.
- En DIR1, el motor aleja los "runners" hasta un máximo de 180 mm respecto a la posición de reposo.

En la siguiente figura se detalla la PCB VAI 4.0.



Manual 1: Detalle PCB VAI 4.0.

1) Conector programación/depuración de la PCB con el PICkit3.
2) Botón de reset.
3) Agujeros de fijación.
4) Botón On/Off. Activa/desactiva la acción del módulo VAI.
5) Conector DB9 para el bus CAN.
6) Interruptor de la alimentación del motor.
7) Conector de la fuente de alimentación del motor.
8) Conector del actuado lineal paso a paso. Pines, 1 +Vm, 2 Q1, 3 Q4, 4 Q2 y 5 Q3.

9) Conector del ATS616, para medir rpm. Pin 1 +5V
10) Conector del Sensor Fin de Carrera.
11) Conector de la alimentación del microcontrolador.
12) Interruptor de la alimentación del microcontrolador.
13) Conector del TPS. Pin 1 VTPS+
14) Indicador encendido/apagado del módulo.
15) Indicadores del estado del módulo.
16) Indicador de sobre temperatura o corriente.

2. Tablas de mapa motor y TPS mínimo.

La modificación o creación de tablas nuevas se realiza enviando tramas con SID 0x001, según la estructura de enteros siguiente, siendo las rpm los bits menos significativos, de un total de 64 bits de datos.

SID1					
RPM	mm	MAXPOSC	NO IMPORTA	extra	orden

Manual 2: SID 0x001.

En el campo extra se debe enviar el número de mapa a modificar o el TPS mínimo nuevo, según la orden enviada. Cualquier dato que no se encuentre en los campos definidos en la figura anterior, son ignorados.

Las especificaciones para la generación de mapas de motor son:

- máximo 6 mapa motor, numerados del 0 al 5.
- máximo 14 posiciones de “runners”. De 1 a 14.
- El rango de rpm válido es de 8000 a 15000 rpm.
- El rango de mm válido es de 0 mm a 180 mm.
- Incremento mínimo de 500 rpm entre posiciones.
- Incremento mínimo de 1 mm entre posiciones.
- La primera posición es 0 mm a 8000 rpm.

Las especificaciones para el cambio de TPS mínimo son:

- El rango de TPS valido es de 1% a 100%.

El posicionado de los “runners” se realiza según las siguientes normas:

- Se alejan de la posición de reposo cuando el TPS esta por encima del TPS mínimo. Y las rpm son mayores que las de la posición actual del motor (DIR1).
- Se acercan a la posición de reposo siempre que las rpm bajen de lo indicado en la posición anterior, sin importar el TPS (DIR0).

3. Cambio de mapa motor.

Para el cambio de mapa motor los datos ocupan los 16 bits más altos de un total de 64 bits de datos, la campo de datos de estas tramas SID2 debe ser:

SID2		
NO IMPORTA	GEAR	orden

Manual 3: SID 0x002.

La orden enviada debe llevar el número adecuado y en el campo GEAR, se debe enviar el número de mapa motor al que se desea cambiar. Si esta trama se envía cada vez que se cambia de marcha en el monoplaza, se dispondrá de un mapa motor para cada marcha. Si se envía mas información en este tipo de trama el calculador VAI, la ignora sin producirse ningún efecto.

4. Órdenes por bus CAN.

En la siguiente tabla se muestran los números de orden que producirán alguna respuesta por parte del módulo, cualquier orden enviada con número y SID no definidas según la tabla no produce ningún efecto.

ID mensaje	ORDEN	
SID1 = 0x001	NTABLA	0x00
	MTABLA	0x01
	MTPS	0x04
SID2 = 0x002	RSTCAN	0x02
	VAISW	0x03
	CHMAPA	0x05

Manual 4: Órdenes VAI CAN.

Según el SID del mensaje se espera un tipo de datos u otro. Según lo expuesto en los puntos 2 y 3. La definición de cada orden es:

- *NTABLA:* Para la creación de una nueva tabla, cuando se crea una tabla, todos los mensajes deben llevar este número de orden, enviándose según sus normas de formación de tabla, uno detrás de otro y tardando menos de 2 minutos entre mensaje y mensaje.
- *MTABLA:* Modifica una posición en una tabla ya creada, se deben cumplir las normas de la creación de tablas, para que la modificación se realice correctamente.
- *MTPS:* Cambia el valor del TPS mínimo para que la separación entre "runners" y la posición de reposo aumente.
- *RSTCAN:* Realiza un reset software del módulo VAI.
- *VAISW:* Con esta orden se activa o desactiva la acción del módulo VAI, quedando los "runners" en la posición que estuvieran si se desactiva el módulo y cuando se activa recupera el funcionamiento normal.
- *CHMAPA:* Orden para cambiar el mapa motor, es decir, cambiar la tabla en uso.

5. Datos del módulo VAI.

El módulo VAI envía periódicamente cada 14.5 ms una trama de datos con las rpm actuales, la posición del TPS actual, distancia en mm entre los "runners" y la posición de reposo, el número de mapa actual, en el campo extra hay información adicional del módulo y en orden se envía el estado del módulo. Los datos son un total de 64 bits, siendo los bits menos significativos los correspondientes a las revoluciones por minuto del monoplaza. Las tramas enviadas por el módulo se envían con SID 0x003 y la organización de los datos se ve en la figura siguiente:

DATOTX								
RPM	TPH	mm	n_mapa	vacio	extra	vacio	orden	vacio

Manual 5: Datos VAI, SID 0x003.

La respuesta del módulo ante una orden recibida será 5 o 6 dependiendo del SID de la trama recibida y junto con el campo extra tienen la siguiente forma:

EXTRA		ORDEN
Info	Ovf	5,6
0,1,2,3	0,1	

Manual 6: Uso 1 del campo extra.

- ***Info:*** Según su valor indica:
  - ***0.*** Error, orden no ejecutada.
  - ***1.*** Esperando una nueva posición al formar una tabla nueva.
  - ***2.*** Error en los datos.
  - ***3.*** Orden realizada.
- ***Ovf:*** Un 1 indica que se produjo overflow en alguna FIFO de recepción.

En el envío de cualquier otra orden el campo extra de datos notifica del estado de reset del módulo:

EXTRA			
RSTMotor	RSTTabla	RSTSensor	RST

Manual 7: Uso 2 del campo extra.

Los tres bits menos significativos cuando alguno esta en '1' indica que se realizará el reset de esa parte del módulo y '0' cuando el reset ya se realizó. Cuando RST es '1' indica que el motor paso a paso esta desplazando los "runners" a la posición de reposo, quedando en '0' cuando alcanza dicha posición.

8. ANEXO II.

CÓDIGO SOFTWARE VAI_v4.0

```
/*
 * File:   VAI_15.c
 * Author: James
 *
 * Created on 30 de enero de 2013, 13:17
 *
 * Hay 6 mapas de motor distintos, pero pueden habilitarse mas
 *
 * Mejora del CAN, en las contestaciones ante las distintas ordenes que puede recibir
 * con dos modos de estado, en escucha y creacion de tabla y funcionando, con
 * distintos codigos de estado para notificar al bus el estado del calculador
 * Opcion de reset por CAN, apagar VAI, modificar una posicion y cambiar el
 * nivel de TPH, cambio de mapa motor y notificacion del nº mapa en uso.
 * Combinando CHMAPA y extra (nº marcha actual del coche) con el n_mapa, el
 * n_mapa variara con el nº marcha
 *
 * Lecturas del ADC, IC1  correctas
 * El motor se mueve bien dentro del numero de posiciones, salto de HOLD a HOLD,
 * cambia de direccion, inc/dec varias posiciones a la vez.
 * Realizado perifil de velocidad trapezoidal, velocidad 6.04mm/s
 * Mejora con la transion de estados para el control del motor mediante la funcion
 * main y el timer4.
 * Unico estado de RESET, tras el cual el motor esta en posicion de inicio se ha
 * iniciado el ADC y el OC5 e introducido una tabla de memoria.
 */

/*1. LIBRERIAS */
#include <stdio.h>
#include <stdlib.h>
#include "p32xxxx.h"
#include "GenericTypeDefs.h"
#include "CAN.h"
#include "plib.h"

/* 2. DEFINICION DE CONSTANTES */
#define IncDistMotor     1.016        /* dist min retornada por el drv880 (Portescap) */
#define OFFSETmm   2*IncDistMotor     /* mm entre la posicion 0 y el sensor FC */
#define ESPERA     0.0145*8276        /* espera entre posiciones tabla nueva */
#define VREF       334                /* Vref_ext con decimales quitados 3.34V*/
#define RPMMIN     8000               /* rpm min para actuar */
#define MAXMAPAS   6                  /* max numero de mapas motor*/
#define MAXTABLA   14                 /* 70 posiciones */
// Estados del automata
#define nFAULT  0   /* estdo overtemp */
#define RESET   1   /* estado reset */
#define HOLD    2   /* mantiene poicion */
#define DIR1    3   /* direccion creciente */
#define DIR0    4   /* direccion decrecinte */
#define WAIT    5   /* estado de espera */
#define RECEP   6   /* recibiendo tabla */
#define ERROR   7   /* estado de error*/
#define OFF     8   /* estado apgado VAI*/
#define CONFIG  9   /* el motor sigue las rpm y THP */
#define CALIBRA 10  /* calibrado inicial de los runners */
// SID de mensajes
#define SID1 0x001      /*SID*/
#define SID2 0x002      /*SID*/
#define SID3 0x003      /*SID*/
#define MAXTX 0x8       /*nº de octetos a enviar*/
// Ordenes
#define NTABLA  0   /* crear tabla nueva */
#define MTABLA  1   /* modificar una posicion */
#define RSTCAN  2   /* reset producido por CAN */
#define VAISW   3   /* enciende apaga el VAI */
#define MTPH    4   /* modifica el valor de TPHMIN */
#define CHMAPA  5   /* cambia a otro mapa motor */
// Estructura de la tabla
typedef union{
    struct{
        unsigned rpm:16;     /* rpm salto a siguiente posicion */
        unsigned mm:8;       /* distancia en mm de la posicon */
        unsigned maxposc:8; /* maximo nº de posiciones */
    };
    UINT32 messageWord;
}TABLA;
// Estructura del dato a enviar
typedef union{
    struct{
        unsigned rpm:16;     /* rpm actuales */
        unsigned TPH:8;      /* poscion en % del TPH actual*/
        unsigned mm:8;       /* distancia actual */
        unsigned n_mapa:4;   /* nº mapa en uso */
        unsigned :12;
        unsigned status:4;   /* si se introduce algun mensaje adicional */
        unsigned :4;
        unsigned estado:4;   /* estado actual del calculador */
        unsigned :4;
    };
    UINT messageWord[2];
}DATOTX;
// Estructura del dato recibido
typedef union{
    struct{
        unsigned :32;        /* vacio CAN */
        unsigned :32;        /* vacio CAN*/
        unsigned rpm:16;     /* rpm nuevas */
        unsigned mm:8;       /* distancia nueva*/
        unsigned maxposc:8;  /* posiciones maximas o posicion a cambiar */
        unsigned :16;        /* vacio CAN */
        unsigned extra:8;    /* orden de recepcion */
```

```
        unsigned orden:8;    /* informacion adicional a la orden */
    };
    UINT32 messageWord[4];
}DATORX;
// Estructura del dato recibido
typedef union{
    struct{
        unsigned rpm:16;     /* rpm nuevas */
        unsigned mm:8;       /* distancia nueva*/
        unsigned maxposc:8;  /* posiciones maximas o posicion a cambiar */
        unsigned orden:8;    /* orden de recepcion */
        unsigned extra:8;    /* informacion adicional a la orden */
    };
    UINT32 messageWord[4];
}InfoCAN;
// Estructura del reset
typedef union{
    struct{
    unsigned RSTMotor:1;     /* TRUE resetea motor a inicio */
    unsigned RSTTabla:1;     /* TRUE carga la tabla de memoria */
    unsigned RSTSensor:1;       /* TRUE reset interno inicio captadores */
    unsigned RST:1;          /* TRUE se esta situando el motor en posicion 0 */
    };
    unsigned status:4;       /* status */
}RST;
// Estructura de estado CANTX
typedef union{
    struct{
    unsigned info:3;         /* TRUE resetea motor a inicio */
    unsigned ovf:1;          /* TRUE carga la tabla de memoria */
    };
    unsigned status:4;       /* status */
}CANst;
/* 3. CONFIGURATION BITS */
// Config settings
// POSCMOD = OFF, FNOSC = FRCDIV, FWDTEN = OFF
// PLLIDIV = DIV_2, PLLMUL = MUL_20
// PBDIV = 8 (default)
// Main clock = 8MHz /2 * 20    = 80MHz
// Peripheral clock = 80MHz /8  =  10MHz

// Configuration Bit settings
// SYSCLK = 80 MHz (8MHz Crystal/ FPLLIDIV * FPLLMUL / FPLLODIV)
// PBCLK = 10 MHz
// FRC Osc FRCPLL
// WDT OFF
// Other options are don't care
//
#pragma config FWDTEN = OFF      /*deshabilito el WDT*/
/*Programacion del oscilador Fcpu=80MHz*/
#pragma config POSCMOD = OFF     /*deshabilita el oscilador primario*/
#pragma config FSOSCEN = OFF     /*deshabilita el osciclador secundario*/
#pragma config FNOSC = FRCPLL    /*FRC 8MHz PLL*/
#pragma config FPLLODIV = DIV_1  /*out divPLL*/
#pragma config FPLLIDIV = DIV_2  /*input divPLL*/
#pragma config FPLLMUL = MUL_20  /*mult PLL*/
/*Progamacion del Fph=10MHz*/
#pragma config FPBDIV = DIV_8    /*div frec perif*/

/* 4. VARIABLES GLOBALES */
const int __attribute__ ((space(prog))) PerfilPR[40] = {0xC350, 0xC350, 0xC350, 0xC350, 0xC350, 0xC350, 0xC350, 0xC350,
                                        0x9C40, 0x9C40, 0x9C40, 0x9C40, 0x9C40, 0x8E0B, 0x8E0B, 0x8E0B,
                                        0x8E0B, 0x8E0B, 0x8E0B, 0x8E0B, 0x8E0B, 0x8E0B, 0x8E0B, 0x8E0B,
                                        0x8E0B, 0x8E0B, 0x8E0B, 0x9C40, 0x9C40, 0x9C40, 0x9C40, 0x9C40,
                                        0xC350, 0xC350, 0xC350, 0xC350, 0xC350, 0xC350, 0xC350, 0xC350};     /* perfil de velocidad */
const UINT __attribute__ ((space(prog))) ADDRESS_TPH = 0x9D008000 ;       /* direccion de almacenamiento virtual TPHMIN */
const UINT __attribute__ ((space(prog))) ADDRESS_TABLA0 = 0x9D009000 ;    /* direccion de almacenamiento virtual ADDRESS_TABLA0 */
const UINT __attribute__ ((space(prog))) ADDRESS_TABLA1 = 0x9D00A000 ;    /* direccion de almacenamiento virtual ADDRESS_TABLA1 */
const UINT __attribute__ ((space(prog))) ADDRESS_TABLA2 = 0x9D00B000 ;    /* direccion de almacenamiento virtual ADDRESS_TABLA2 */
const UINT __attribute__ ((space(prog))) ADDRESS_TABLA3 = 0x9D00C000 ;    /* direccion de almacenamiento virtual ADDRESS_TABLA3 */
const UINT __attribute__ ((space(prog))) ADDRESS_TABLA4 = 0x9D00D000 ;    /* direccion de almacenamiento virtual ADDRESS_TABLA4 */
const UINT __attribute__ ((space(prog))) ADDRESS_TABLA5 = 0x9D00E000 ;    /* direccion de almacenamiento virtual ADDRESS_TABLA5 */
UINT __attribute__ ((space(data))) TPHMIN;                        /* posicion del minimo TPH en % */
TABLA __attribute__ ((space(data))) ListaPosicion[ MAXTABLA];     /* tabla de lista de posiciones */
#ifdef debug
TABLA __attribute__((space(data))) ListaPosicion0[MAXTABLA] = {{0x1F40,0x00,0x06}, {0x2134,0x10,0x06}, {0x2328,0x14,0x06}, {0x2AF8,0x18,0x06},
                                                               {0x2FDA,0x42,0x06}, {0x3A98,0xB4,0x06}};    /* tabla de lista de posiciones */
TABLA __attribute__((space(data))) ListaPosicion1[MAXTABLA] = {{0x1F40,0x00,0x03}, {0x2134,0x10,0x03},
                                                               {0x2FDA,0x42,0x03}};    /* tabla de lista de posiciones */
TABLA __attribute__((space(data))) ListaPosicion2[MAXTABLA] = {{0x1F40,0x00,0x03}, {0x2FDA,0xB4,0x06},
                                                               {0x3A98,0xB4,0x06}};    /* tabla de lista de posiciones */
#endif
int ESTADO = CALIBRA;     /* estado del calculador (estado inicial)*/
int POSACTUAL = 0;        /* posicion del motor */
int POSFIN  = 0;          /* posicion destino del motor */
int MAXPOSC = 6;          /* nº maximo de posiciones */
int Indice = 0;           /* indice del perfil de velocidad*/
int Espera = 0;           /* indice de cuenta espera entre posciones*/
float DistActual = 0;     /* indice para lectura de posiciones */
BOOL InterruptorVAI = FALSE;     /* interruptor VAI. EXT1, CANRx */
RST RSTStatus = {TRUE, TRUE, TRUE};     /* estado de reset del modulo, tabla y ADC señalados */
CANst CANStatus = {0,0};    /* ovf y codigo de respuesta */
// ADC
UINT *ADCptr;             /* puntero a buffer de conversion */
//CAN
unsigned int CANFifoMensajeBuffer[120];   /* buffer con nº total del words en las FIFOs */
DATORX CANBufferRx;      /* buffer de informacion de recepcion del modulo CAN */
DATOTX CANBufferTx;       /* buffer de escritura del modulo CAN */
BOOL transmitir = FALSE; /* acciona la transmision */
```

```
// IC
UINT *IC1ptr;           /* puntero a buffer de IC */
BOOL IC1Event = FALSE;  /* testigo de que hay una medida de frec *///UINT flanco1=0, flanco2=0;  /* variables para medir frecuencia */
UINT IC1Buffer[3];      /* IC1 buffer para almacenar flancos */

/* 5. PROTOTIPOS DE FUNCIONES */
extern void setup_micro  (void);          /*inicializacion del micro*/
extern void CAN_config(void);             /*configura la comunicacion CAN*/
extern BOOL CAN_transmit(DATOTX *dato); /* transmite un dato por CAN*/
extern CAN_EVENT_CODE CAN_recive(DATORX *dato, CAN_MODULE_EVENT modEvent);    /* recive un dato por CAN*/
extern void ErrorBusEvent  (void);        /* gestiona estados de error del bus CAN */
extern void SystemErrorEvent  (void);     /* gestiona errores del sistema CAN */
extern BOOL RxOverflowEvent  (void);      /* gestiona overflow en FIFO RX */
extern int CreaTabla  (TABLA *tabla);     /* Crea la tabla de posiciones */
extern UINT GuardaTabla(TABLA *tabla, const UINT memWrite); /* guarda tabla en Flash */
extern UINT CargaTabla(TABLA *tabla, const UINT memRead);   /* carga tabla de la Flash */
extern UINT GuardaTPH(const UINT memWrite);                 /* guarda TPHMIN en Flash */
extern UINT CargaTPH(UINT *TPH, const UINT memRead);        /* carga TPHMIN de la Flash */
extern BOOL ModificaTabla(TABLA *tabla);/* modifica una posicion de la tabla */
extern BOOL ControlMotor  (int estado);   /* gestiona el control del motorDC */
void MedirRPM_TPH ( void);                /* gestiona la medida y calculo de rpm */
void GestCANRx ( void);                   /* gestiona la recepcion CAN */
void GestConfig ( BOOL *contesta);        /* gestiona la rececpion de config */
void GestRst ( void);                     /* gestiona el reinicio del calculador */
void GestWait(BOOL *contesta);            /* gestiona el estado de espera del CAN */
void GestMotor ( void);                   /* gestiona el movimiento del motor */
void GestCANTx ( void);                   /* gestiona el envio de datos rpm, TPH al bus */
void GestFAULT ( void);                   /* gestina sobretemperatura en DRV8805 */
void GestError ( void);                   /* gestiona errores en el CAN y ovf */
void GestInterruptor ( void);             /* gestiona el interruptor interno */

/*
 * Control del calculador VAI
 *
 * gestion la acion del modulo del estado actual.
 */
void main(void) {
    BOOL contesta = FALSE;
    BOOL *ptrSW;

    /*funciones de configuracion*/
    INTEnableSystemMultiVectoredInt();    /* Activa el modo Multi Vector */
    setup_micro();                        /* inicializo el micro */
    CANEnableModule(CAN1, TRUE);          /* habilito el modulo CAN */
    CAN_config();                         /* configura el CAN */
#ifdef debug
    GuardaTabla(ListaPosicion0, ADDRESS_TABLA0); // ESTA AQUI PARA PRUEBAS, al quitar esta lineas la primera vez la memoria estara vacia
    CANBufferRx.extra = 65;
    GuardaTPH(ADDRESS_TPH);             // hay que introducir al menos una lista por bus can
    GuardaTabla(ListaPosicion1, ADDRESS_TABLA1);
    GuardaTabla(ListaPosicion2, ADDRESS_TABLA2);
#endif
    CANBufferTx.n_mapa = 0;               /* mapa inicial */
    ptrSW = &contesta;                    /* inicio puntero */

    while (ESTADO == CALIBRA){
        if(RSTStatus.RSTMotor){
            RSTStatus.RST = TRUE;
            CANBufferTx.status = RSTStatus.status;  /* estado del reset RSTstatus */
            DistActual = 0;                         /* posicion actual */
            POSACTUAL = 0;                          /* posicion minima */
            POSFIN = 0;                             /* posicion minima */
            ControlMotor( RESET);                   /* resetea drv8805 y mueve motor */
            RSTStatus.RSTMotor = FALSE;             /* situa el motor a una distancia offset del SFC */
        }
    }
    while(1){
        CANBufferTx.estado = ESTADO;  /* carga el estado actual */
        switch (ESTADO){     /* gestiona todo el calculador */
            case CONFIG:     /* cofigura tablas y TPHMIN */
                GestConfig(ptrSW);
                break;
            case RECEP:      /* controla las recepciones CAN */
                GestCANRx();
                break;
            case nFAULT:     /* estado de fallo en motor (DRV8805) */
                GestFAULT();
                break;
            case RESET:      /* estado de reinicio del calculador */
                GestRst();
                break;
            case WAIT:       /* estado de espera */
                GestWait(ptrSW);
                break;
            case ERROR:      /* estado de error */
                GestError();
                break;
            case HOLD:       /* mueve el motor si se alcanzo la posicion */
                GestMotor();
                break;
            default:

                break;
        }
        MedirRPM_TPH();      /* calcula las rpm */
        GestCANTx();         /* envia los datos al bus */
    }
}
/*
 * EXT2 interrupt service rutine
```

```
 *
 * Pulsador para apagar el VAI, con lo que el motor sequedara quieto donde este
 * aunque cambien rpm y TPH.
 */
void __ISR(_EXTERNAL_2_VECTOR, ipl2) EXT2ResetPosicion ( void){
    IFS0CLR = 0x00000800;    /* Bajo la bandera de la int externa 2 */

    GestInterruptor();       /* gestiona interruptor */

}
/*
 * EXT3 interrupt sevice rutine
 *
 * Atiende al sensor fin de carrera (SFC), serian 2 pulsadores conectados
 * a la misma interrupcion, sensor inferior y sensor superior.
 * Segun el estado del automanta realiza una accion.
 */
void __ISR(_EXTERNAL_3_VECTOR, ipl2) EXT3SensorCarrera ( void){
    IFS0CLR = 0x00008000;    /* Bajo la bandera de la int externa 3 */
    PORTDSET = 0x20;         /* reset drv8805 */

    switch (ESTADO){
        case DIR1:
            TMR4 = 0xFFFF;          /* distancia max de la cuenta de "vueltas" */
            DistActual = 180 +  OFFSETmm;      /* posicion extra maxima */
            POSACTUAL = MAXPOSC-1; /* posicion maxima */
            POSFIN = MAXPOSC-1;    /* posicion maxima */
            RSTStatus. RST = TRUE;/* indica situacion */
            ESTADO =  CALIBRA;       /* en espera alcanza ultima posicion */
            ControlMotor( DIR0);   /* cambio de direccion y estado */
             break;
        case DIR0:
            TMR4 = 0;               /* distancia min de la cuenta de "vueltas" */
            DistActual = 0;         /* posicion actual */
            POSACTUAL = 0;          /* posicion minima */
            POSFIN = 0;             /* posicion minima */
            RSTStatus. RST = TRUE;/* indica situacion */
            ESTADO =  CALIBRA;       /* en espera alcanza posicion inicial */
            ControlMotor( DIR1);   /* cambio de direccion y estado */
             break;
        case RESET:
            TMR4 = 0;               /* distancia min de la cuenta de "vueltas" */
            DistActual = 0;         /* posicion actual */
            POSACTUAL = 0;          /* posicion minima */
            POSFIN = 0;             /* posicion minima */
            ESTADO =  CALIBRA;       /* en espera alcanza posicion inicial */
            ControlMotor( DIR1);   /* cambio de direccion y estado */
             break;
        case CALIBRA:
            TMR4 = 0;               /* distancia min de la cuenta de "vueltas" */
            DistActual = 0;         /* posicion actual */
            POSACTUAL = 0;          /* posicion minima */
            POSFIN = 0;             /* posicion minima */
            ESTADO =  CALIBRA;       /* en espera alcanza posicion inicial */
            ControlMotor( DIR1);   /* cambio de direccion y estado */
             break;
        default:
             break;
    }
}

/*
 * CAN interrupt service routine
 *
 * Se gestionan las distintas fuentes de interrupcion del bus CAN, se muestra estado
 * del bus por PORTE
 */
void __ISR(_CAN_1_VECTOR, ipl1) CAN1Interrupt (void){
    CAN_EVENT_CODE  fifo;           /* identificado de la FIFO*/
    CAN_MODULE_EVENT  moduleEvent;
    UINT errorCount;

    errorCount = CANGetRxErrorCount(CAN1);    /* numero de errores en el bus CAN */
    if(errorCount < 200)
    {
        // si la cuenta de error es mayor
        // hacer algun diagnostico
        // Ver especificacion CAN2.0
        // Limpio los leds de error

        PORTECLR = 0x0F;          /* apago leds de estado bus */
    }
    moduleEvent = CANGetModuleEvent(CAN1);    /* evento CAN ocurrido en el calculador */
    if((moduleEvent & (CAN_RX_EVENT | CAN_INVALID_RX_MESSAGE_EVENT)) != 0){
        // se controlan los eventos invalid message y RX module Event
        fifo = CAN_recive(&CANBufferRx, moduleEvent);
        if (fifo == CAN_CHANNEL1_EVENT)
            ESTADO =  CONFIG;     /* estado de configuracion */
        else if (fifo == CAN_CHANNEL2_EVENT)
            ESTADO =  RECEP;      /* estado de recepcion */
    }
    else if((moduleEvent & CAN_SYSTEM_ERROR_EVENT ) != 0){
        // identifico la fuente del error y se actua
        SystemErrorEvent ();
    }
    else if((moduleEvent & CAN_BUS_ERROR_EVENT ) != 0){
        // se activa al subir un nivel de error (warning, passive, Bus OFF)
        ErrorBusEvent ();
    }
    else if((moduleEvent & CAN_RX_OVERFLOW_EVENT ) != 0){
```

```
        // identifica y crea un reporte de perdida de informacion
        if (RxOverflowEvent ())
            CANStatus. ovf = 1;       /* error por overflow de recepcion */
        ESTADO =  ERROR;      /* CAMBIA ESTADO */
    }
    else{
        // interrupcion desconocida
        ESTADO =  ERROR;      /* CAMBIA ESTADO */
    }
    IFS1CLR = 0x04000000;     /* limpia flag CAN1 */
}

/*
 * ADC interrupt service routine
 *
 * El valor leido se pasa % de la tension de referencia, se adquiere la muestra
 * de forma automatica, se da la orden de conversion despues de calcular rpms
 */
void __ISR( _ADC_VECTOR, ipl1) ADCInterrupt ( void){
    static UINT var;          /* variable para el calculo */
    static float tension;     /* almacena la tension actual */

    ADCptr = &ADC1BUF0;       /* inicializa el puntero al buffer de conversion */
    IFS1CLR = 0x00000002;     /* bajo la bandera ADC */

    var = *ADCptr;
    tension = (var)* VREF/1023;  /* calculo del valor leido y desplazo decimales */
    var = tension*100/ VREF;     /* posicon TPS en % */
    CANBufferTx. TPH = var;      /* carga posicion TPS en % */
}
/*
 * Input Capture 1 interrupt sevice routine
 *
 * Se emplea el timer 3 para el numero de ciclos transcurridos entre flanco y flanco
 * de un señal cuadrada, para medir su frecuencia. (ajustado para sensor Hall ATS616
 * que midiendo el motor tiene una sensibilidad de 1Hz/rpm
 */
void __ISR(_INPUT_CAPTURE_1_VECTOR , ipl6) IC1Int( void){
    static int i = 0;

    IC1ptr = &IC1BUF;         /* inicializa el puntero a la FIFO */
    IC1Buffer[i]= *IC1ptr;    /* escribe en buffer IC1 */
    i++;
    if (i>=3){       /* captura tres flancos */
        IC1CONCLR = 0x0007;  /* capture disable mode */
        IC1Event = TRUE;     /* señala evento */
        i=0;                 /* reinicia cuenta */
    }

    IFS0CLR = 0x00000020;     /* bajo la bandera IC1 */
}
/*
 * Timer4 interrupt service routine
 *
 * Desactiva el Output Compare cuando la posicion actual es igual o ha
 * sobrepasado la posicion final. Si se produce un cambio en las rpm y el motor
 * busca de nuevo su posicion.
 * Esta interrupcion esta programada para el motor portescap 57DBM10B2U-L, se
 * produce cada vez que el motor se desplaza 1.016 mm. (error en posicon < 1mm)
 */
void __ISR( _TIMER_4_VECTOR , ipl3) T4Interrupt ( void){
    UINT dist;
    TMR4CLR = 0xffff;         /* valor de recarga timer4 */
    IFS0CLR = 0x00010000;     /* Bajo la bandera de la int timer 4 */
    // CONTROL DE LA POSICION, POR CADA mm

    switch (ESTADO){
        case DIR1:
            DistActual = DistActual +  IncDistMotor;
            dist = DistActual;
            if (ListaPosicion[POSFIN]. mm <= dist){     /* posicion OK */
                OC5CONCLR = 0x8000;       /* disable OC5 */
                POSACTUAL++;              /* incrementa una posicion en la tabla */
                ESTADO =  HOLD;           /* CAMBIA ESTADO */
                ControlMotor(ESTADO);     /* Cambia a hold */

            }
            else if (CANBufferTx. rpm <= ListaPosicion[POSACTUAL]. rpm){     /* debe cambiar direccion */
                if (POSFIN > 0){             /* decrementa una posicion */
                    ESTADO =  DIR0;          /* CAMBIA ESTADO */
                    POSFIN--;                /* intercambia posiciones */
                    POSACTUAL++;             /* intercambia posiciones */
                    ControlMotor(ESTADO);    /* mueve el motor */
                }
            }
            break;
        case DIR0:
            DistActual = DistActual -  IncDistMotor;
            dist = DistActual;
            if (ListaPosicion[POSFIN]. mm >= dist){     /* posicion OK */
                OC5CONCLR = 0x8000;       /* disable OC5 */
                POSACTUAL--;              /* decremea una posicion en la tabla */
                ESTADO =  HOLD;           /* CAMBIA ESTADO */
                ControlMotor(ESTADO);     /* Cambia a hold */
            }
            else if (CANBufferTx. rpm > ListaPosicion[POSACTUAL-1]. rpm){     /* debe cambiar direccion */
                if (CANBufferTx. TPH >= TPHMIN){      /* comprueba TPS */
                    if (POSFIN < MAXPOSC-1){     /* cambia de direccion */
                        ESTADO =  DIR1;          /* CAMBIA ESTADO */
```

```
                    POSFIN++;                   /* intercambia posiciones */
                    POSACTUAL--;                /* intercambia posiciones */
                    ControlMotor(ESTADO);       /* mueve el motor */
                }
            }
        }
        break;
    case RESET:
        if(PORTDbits.RD6 == FALSE){     /* si va en sentido decreciente */
            DistActual = DistActual -   IncDistMotor;
            dist = DistActual;
            if (ListaPosicion[0].mm >= dist){    /* posicion OK */
                RSTStatus.RST = FALSE;
                CANBufferTx.status = RSTStatus.status;
                OC5CONCLR = 0x8000;         /* disable OC5 */
                POSACTUAL = 0;              /* posicion minima */
                POSFIN = 0;                 /* posicion minima */
                ESTADO =  HOLD;             /* CAMBIA ESTADO */
                ControlMotor(ESTADO);       /* Cambia a hold */
            }
        }
        break;
    case CALIBRA:
        if(PORTDbits.RD6 == TRUE){      /* si va en sentido creciente */
            DistActual = DistActual +   IncDistMotor;
            dist = DistActual;
            if (OFFSETmm <= dist){      /* posicion OK */
                RSTStatus.RST = FALSE;
                CANBufferTx.status = RSTStatus.status;
                OC5CONCLR = 0x8000;         /* disable OC5 */
                DistActual = 0;             /* distancia inicio */
                ESTADO =  RESET;            /* CAMBIA ESTADO */
                ControlMotor( HOLD);        /* Cambia a hold */
            }
        }
        else{       /* si va en sentido decreciente */
            DistActual = DistActual -   IncDistMotor;
            dist = DistActual;
            if (ListaPosicion[MAXPOSC-1].mm >= dist){    /* posicion OK */
                RSTStatus.RST = FALSE;
                CANBufferTx.status = RSTStatus.status;
                OC5CONCLR = 0x8000;         /* disable OC5 */
                DistActual = ListaPosicion[MAXPOSC-1].  mm;  /* distancia final */
                ESTADO =  HOLD;             /* CAMBIA ESTADO */
                ControlMotor(ESTADO);       /* Cambia a hold */
            }
        }
        break;
    default:

        break;
    }
    Indice = 0;
    CANBufferTx.mm = DistActual;     /* carga distancia actual mm */
}
/*
 * Timer5 interrupt service routine
 *
 * Indica que se ha de transmitir por el bus CAN
 */
void __ISR( _TIMER_5_VECTOR, ipl3) T5Interrupt( void){
    TMR5CLR = 0x0000;         /* valor de recarga timer5 */
    IFS0CLR = 0x00100000;     /* Bajo la bandera de la int timer 5 */

    transmitir = TRUE;
    if (ESTADO == WAIT){
        Espera++;
        if (Espera == ESPERA){
            Espera = 0;
            RSTStatus.RSTTabla = TRUE;  /* motor a posicion inicial */
            CANStatus.info = 0;   /* error al crear tabla */
        }
    }

}
/*
 * Timer2 interrupt service routine
 *
 * Funcion que se ejecutara 40 veces entre ejecucion y ejecucion del timer 4, carga
 * en los registros el periodo y el ciclo de trabajo para generar el perfil de
 * velocidad.
 */
void __ISR( _TIMER_2_VECTOR, ipl3) T2Interrupt( void){
    IFS0CLR = 0x00000100;     /* Bajo la bandera de la int timer 2 */

    PR2 = PerfilPR[Indice];       /* carga periodo */
    OC5RS = PerfilPR[Indice]/2;  /* carga ciclo de trabajo */
    Indice++;

    if (Indice>39)
        Indice=0;

}
/*
 * Output Compare 5 service routine
 *
 * Detecta el estado nFAULT del drv8805
 */
void __ISR(_OUTPUT_COMPARE_5_VECTOR, ipl4) OC5Interrupt( void){
    IFS0CLR = 0x00400000;     /* Bajo la bandera de la int output compare 5 */
```

```
    ESTADO = nFAULT;        /* se produjo overtemp u overcurrent */
}
/*
 * void MedirRPM_TPH(void)
 *
 * Calcula rpm a partir de la frecuencia medida contando el numero de ciclos del
 * timer 3 transcurridos entre flanco y flanco. Inicia la conversion del ADC
 */
void MedirRPM_TPH ( void){
    float frec;
    long int captura=0; /* numero de cuentas hechas por el timer */

    if (IC1Event){  /* si se activo el evento en el IC1 */
        if (IC1Buffer[2]<IC1Buffer[0])            /* calcula nº ciclos timer transcurridos*/
            captura = IC1Buffer[2]-IC1Buffer[0]+65535;
        else
            captura = IC1Buffer[2]-IC1Buffer[0];
        frec = 1/(captura*0.0000001*1);       /* frecuencia medida */
        captura = frec*2;                     /* calculo de rpm sin decimales y divido entre 0.5Hz/rev */
        CANBufferTx.rpm = captura;            /* carga rpm actuales */
        IC1Event = FALSE;                     /* evento controlado */
        IC1CONSET = 0x0001;                   /* edge detect mode */
        AD1CON1bits.SAMP = 0;                 /* comienza una conversion */
    }
}
/*
 * void GestCANRx( void)
 *
 * Funcion que permite al calculador reconocer la instruccion recibida en un
 * mensaje CAN SID2 y procesarla
 */
void GestCANRx ( void){
    int n_mapa = 0;
    UINT dir;

    ControlMotor(WAIT);     /* Disable output drv8805 */
    switch (CANBufferRx.orden){
        case RSTCAN:     /* reset del modulo */
            RSTStatus.RSTSensor = TRUE;
            RSTStatus.RSTTabla = TRUE;          /* carga status */
            RSTStatus.RSTMotor = FALSE;
            CANStatus.info = 3;                 /* tabla modificada ok */
            CANBufferTx.status = CANStatus.info;/* actualiza status */
            CAN_transmit(&CANBufferTx);         /* transmite CAN */
            ESTADO = RESET;                     /* CAMBIA ESTADO */
            break;
        case VAISW:     /* on/off VAI */
            GestInterruptor();                  /* gestiona interruptor */
            CANStatus.info = 3;                 /* interruptor pulsado */
            CANBufferTx.status = CANStatus.info;/* actualiza status */
            CAN_transmit(&CANBufferTx);          /* transmite CAN */
            break;
        case CHMAPA:    /* cambia a otro mapa motor*/
            n_mapa = CANBufferRx.extra;     /* se emplea extra para indicar nº mapa */
            if (n_mapa<MAXMAPAS ){
                if (n_mapa != CANBufferTx.n_mapa){
                    dir = ADDRESS_TABLA0 +(n_mapa*0x1000);    /* obtiene direccion */
                    CargaTabla(ListaPosicion, dir);           /* recupera tabla de la Flash */
                    RSTStatus.RSTMotor = TRUE;                /* motor a posicion inicial */
                    CANStatus.info = 3;                       /* cambio de mapa ok */
                    CANBufferTx.n_mapa = n_mapa;              /* actualiza e nº mapa actual */
                    ESTADO = RESET;                           /* CAMBIA ESTADO */
                }
                else{   /* no cambia de mapa */
                    CANStatus.info = 0;                       /* error al cambiar mapa */
                    ESTADO = HOLD;                            /* CAMBIA ESTADO */
                }
            }
            else{   /* no cambia de mapa */
                CANStatus.info = 0;                     /* error al cambiar mapa */
                ESTADO = HOLD;                          /* CAMBIA ESTADO */
            }
            CANBufferTx.status = CANStatus.info;
            CAN_transmit(&CANBufferTx);             /* transmite CAN */
            break;
        default:
            // si recibe algo que no esta dentro de ordenes no hace nada
            if (POSFIN == POSACTUAL){    /* recupera el estado anterior */
            ESTADO = HOLD;                  /* CAMBIA ESTADO */
            }
            else if (POSFIN > POSACTUAL){
                ESTADO = DIR1;              /* CAMBIA ESTADO */
            }
            else{
                ESTADO = DIR0;              /* CAMBIA ESTADO */
            }
            ControlMotor(ESTADO);           /* mueve el motor */
            break;
    }
    CANBufferTx.status = RSTStatus.status;      /* al salir actualizo estatus */
}
/*
 * void GestConfig( BOOL *contesta)
 *
 * Funcion que permite al calculador reconocer la instruccion recibida en un
 * mensaje CAN SID1 y procesarla
 */
void GestConfig ( BOOL *contesta){
    ControlMotor(WAIT);     /* Disable output drv8805 */
```

```
    switch (CANBufferRx.orden){
        case NTABLA:        /* crear una tabla */
            switch(CreaTabla(ListaPosicion)){
                case 0:     /* tabla en creacion, posic OK */
                    CANStatus.info = 1;    /* tabla en creacion */
                    *contesta = TRUE;
                    ESTADO = WAIT;         /* CAMBIA ESTADO */
                    break;
                case 1:     /* tabla completa */
                    PORTESET = ListaPosicion[0].maxposc;
                    RSTStatus.RSTMotor = TRUE;  /* motor a posicion inicial */
                    CANStatus.info = 3;    /* tabla creada ok */
                    ESTADO = WAIT;         /* CAMBIA ESTADO */
                    break;
                case 2:     /* deshecha tabla nueva */
                    RSTStatus.RSTMotor = TRUE;  /* motor a posicion inicial */
                    CANStatus.info = 0;    /* error al crear tabla */
                    ESTADO = WAIT;         /* CAMBIA ESTADO */
                    break;

                case 3:     /* error en la creacion posic NOK */
                    CANStatus.info = 2;    /* error al crear tabla */
                    *contesta = TRUE;
                    ESTADO = WAIT;         /* CAMBIA ESTADO */
                    break;
                default:
                    break;
            }
            break;
        case MTABLA:     /* modificar una posicion en la tabla */
            if(ModificaTabla(ListaPosicion))      /* si la tabla se modifico */
                CANStatus.info = 3;        /* tabla modificada ok */
            else
                CANStatus.info = 0;        /* error al modificar tabla */
            RSTStatus.RSTMotor = TRUE; /* motor a posicion inicial */
            ESTADO = WAIT;             /* CAMBIA ESTADO */
            break;
        case MTPH:      /* cambia valor de TPHMIN */
            if (GuardaTPH(ADDRESS_TPH)){     /* si el valor de tph esta en rango*/
                TPHMIN = CANBufferRx.extra;     /* se emplea extra para almacenar TPH */
                CANStatus.info = 3;             /* TPSMIN nuevo ok */
            }
            else
                CANStatus.info = 0;             /* error TPS nuevo */
            RSTStatus.RSTTabla = TRUE;       /* reset de TPS */
            ESTADO = WAIT;                   /* CAMBIA ESTADO */
            break;
        default:
            // si recibe algo que no esta dentro de ordenes no hace nada
            if (POSFIN == POSACTUAL){    /* recupera el estado anterior */
            ESTADO = HOLD;                   /* CAMBIA ESTADO */
            }
            else if (POSFIN > POSACTUAL){
                ESTADO = DIR1;               /* CAMBIA ESTADO */
            }
            else{
                ESTADO = DIR0;               /* CAMBIA ESTADO */
            }
            ControlMotor(ESTADO);            /* mueve el motor */
            break;
    }
}
/*
 * void GestRst( void)
 *
 * Funcion que realiza el reset del calculador, calcula nuevo offset del ADC,
 * extre la tabla de memoria no volatil y lleva el motor a posicion inicial
 * indicando al bus el proceso del reset.
 */
void GestRst ( void){
    int dir;      /* direccion memoria Flash */


    if(RSTStatus.RSTSensor){
        CANBufferTx.status = RSTStatus.status;  /* estado del reset RSTstatus */
        RSTStatus.RSTSensor = FALSE;
        AD1CON1SET = 0x8000;                /* modulo ADC ON */
        IC1CONSET = 0x8001;                 /* IC1 ON, edge detect mode */
        transmitir = TRUE;
    }
    else if(RSTStatus.RSTMotor){
        if (DistActual != 0){
            RSTStatus.RST = TRUE;
            CANBufferTx.status = RSTStatus.status;  /* estado del reset RSTstatus */
            ControlMotor( DIR0);              /* mueve motor */
        }
        else
            ESTADO = HOLD;
        POSACTUAL = 0;         /* posicion minima */
        POSFIN = 0;            /* posicion minima */
        RSTStatus.RSTMotor = FALSE;          /* situa el motor a una distancia offset del SFC */
        transmitir = TRUE;
    }
    else if(RSTStatus.RSTTabla){
        CANBufferTx.status = RSTStatus.status;  /* estado del reset RSTstatus */
        dir = ADDRESS_TABLA0+(CANBufferTx.n_mapa*0x1000); /* obtiene direccion */
        CargaTabla(ListaPosicion, dir);       /* recupera tabla anterior de la Flash */
        CargaTPH(&TPHMIN, ADDRESS_TPH);       /* carga TPH min de memoria */
        RSTStatus.RSTTabla = FALSE;          /* CAN tabla cargada */
```

```
        RSTStatus.RSTMotor = TRUE;          /* reset del motor */
        InterruptorVAI  = FALSE;            /* config por defecto del sw */
        transmitir = TRUE;
    }
}
/*
 * void GestWait(BOOL *contesta)
 *
 * En periodo de espera de construccion de una nueva tabla indica si los datos
 * introducidos fueron correctos, erroneos o si se ha finalizado bien la tabla
 * o si se recupero la anterior de memoria no volatil
 */
void GestWait(BOOL *contesta){

    if((CANStatus.info == 1)||(CANStatus.info == 2)){      /* tabla en creacion */
        if (*contesta){
            CANBufferTx.status = CANStatus.info;
            CAN_transmit(&CANBufferTx);   /* transmite CAN */
            ESTADO =  WAIT;               /* CAMBIA ESTADO */
            Espera = 0;                   /* inicia el periodo de espera */
            *contesta = FALSE;
        }
    }
    else{             /* tabla creada ok */
        CANBufferTx.status = CANStatus.info;
        CAN_transmit(&CANBufferTx);   /* transmite CAN */
        ESTADO =  RESET;              /* CAMBIA ESTADO */
        CANBufferTx.status = RSTStatus.status;
    }
}
/*
 * void GestMotor( void)
 *
 * Controla que el motor busque su posicion adecuada arrancando su marcha en la
 * direccion adecuada
 */
void GestMotor ( void){
    if (CANBufferTx.rpm > ListaPosicion[POSACTUAL].rpm){    /* umbral max */
        if (CANBufferTx.TPH >= TPHMIN){  /* comprueba TPH */
            if (POSFIN < MAXPOSC-1){     /* incrementa una posicion */
                ESTADO =  DIR1;          /* CAMBIA ESTADO */
                POSFIN++;                /* posicion final */
                ControlMotor(ESTADO);    /* mueve el motor */
            }
        }
    }
    else if (POSACTUAL > 0){     /* comprueba no esta en la primera posicion*/
        if (CANBufferTx.rpm <= ListaPosicion[POSACTUAL-1].rpm){    /* umbral min */
            if (POSFIN > 0){             /* decrementa una posicion */
                ESTADO =  DIR0;          /* CAMBIA ESTADO */
                POSFIN--;                /* posicion final */
                ControlMotor(ESTADO);    /* mueve el motor */
            }
        }
    }
}
/*
 * void GestCANTx( void)
 *
 * Funcion para el envio de rpm TPH y status del VAI
 */
void GestCANTx ( void){

    if ((ESTADO!=WAIT) && (ESTADO!=RECEP) && (ESTADO!=CONFIG)){
        if(transmitir){      /* transmite las rpm, mm y el TPH */
            CANBufferTx.estado = ESTADO;            /* carga el estado actual */
            CAN_transmit(&CANBufferTx);             /* transmite CAN */
            CANBufferTx.status = RSTStatus.status;  /* estado del reset RSTstatus */
            transmitir = FALSE;                     /* se limpia el evento */
        }
    }
}
/*
 * void GestFAULT( void)
 *
 * Gestiona el estado nFAULT, se sondea para reanudar el motor
 */
void GestFAULT ( void){

    if (ControlMotor(ESTADO)){   /* si sale de nFAULT */
        ESTADO =  RESET;                /* CAMBIA ESTADO */
        RSTStatus.RSTMotor = TRUE;      /* reset del motor */
    }
    transmitir = TRUE;
}
/*
 * void GestError( void)
 *
 * Controla los errores en el bus CAN del modulo, como solo recibe ordenes y la
 * nueva tabla, ante un ovf y su perdida de informacion, se reseteara el calculador
 */
void GestError ( void){

    if (CANStatus.ovf = 1){  /* se detecto overflow */
        CANBufferTx.status = CANStatus.ovf; /* señala estado */
        CAN_transmit(&CANBufferTx);         /* transmite CAN */
        CANStatus.ovf = 0;                  /* reinicio error por overflow */
    }
    else{   /* error en el CAN del calculador */
        CANBufferTx.status = 0xF;           /* error en el CAN del calculador */
```

```
        CAN_transmit(&CANBufferTx);           /* transmite CAN */
    }
    RSTStatus.RSTSensor  = TRUE;
    RSTStatus.RSTTabla  = TRUE;           /* reset del calculador*/
    RSTStatus.RSTMotor  = FALSE;
    ESTADO =  RESET;                      /* CAMBIA ESTADO */
}
/*
 * void GestInterruptor( void)
 *
 * gestiona la desconexion del sistema VAI, cuando se reactiva, recupera el
 * anterior de movimiento del PaP.
 */
void GestInterruptor ( void){

    if (!InterruptorVAI){
        ControlMotor( WAIT);      /* Disable output drv8805 */
        ESTADO =  OFF;            /* VAI desactivado */
        InterruptorVAI = TRUE;
    }
    else{
        InterruptorVAI  = FALSE;
        if (POSFIN == POSACTUAL){    /* recupera el estado anterior */
        ESTADO =  HOLD;                   /* CAMBIA ESTADO */
        }
        else if (POSFIN > POSACTUAL){
            ESTADO =  DIR1;               /* CAMBIA ESTADO */
        }
        else{
            ESTADO =  DIR0;               /* CAMBIA ESTADO */
        }
        ControlMotor(ESTADO);             /* mueve el motor */
    }
}
```

```
/*********************************************************************/
/*  Creación:                                                        */
/*    · Jaime García Padilla                                         */
/*********************************************************************/
/*  devprogVAI_14.c                                                  */
/*  Fichero con las rutinas de inicialización                        */
/*  Creación:   octubre-2012                                         */
/*  Modificaciones:                                                  */
/*********************************************************************/

#include "p32xxxx.h"
#include "plib.h"

#define PORTESCAP

void setup_micro (void){
    // Deshabilto jtag
    AD1PCFG = 0xFFFF;
    // configura PORTA pins
    TRISA = 0x4600;      // all PORTA as output, pin 66 RA14/INT3, pin 28/Vref-, pin 29/Vref+ input
    // configura PORTB pins
    TRISB=0x8003;        // all PORTB as output, pin 25/RB0/AN0, 24/RB1/AN1 input, 44 RB15/OCFB
    AD1PCFG=0xFFFC;      // all PORTB as digital, pin 25/RB0/AN0, 24/RB1/AN1 analog
    // configura PORTC pins
    TRISC = 0x48;        // pin 74/RC14/T1CK, 8 RC2/T4CK input
    // configura PORTD pins
    TRISD = 0x0100;      // all pins outputs, pin 68/IC1/RD8 input
    PORTD = 0;           // inicializo el puerto
    // configure all PORTE pins
    TRISE = 0xFF0;       // all PORTE as output, int1,int2 input (pin 18,19)
    PORTECLR = 0x00F;    // inicializo el puerto
    // configura PORTF para el CAN1
    TRISFSET=0x0001;     // RF0.0 87 INPUT Schmitt Trigger input with CMOS levels
    TRISFCLR=0x0002;     // RF1 88 OUTPUT

    // configure 10-bitADC
    /*
     * prepara la siguiente muestra automatica, 1 muestras por interrupcion, resultado
     * integer 16 bit, el periodo de muestreo es el periodo de la interrupcion del
     * capture IC1<<periodo de muestreo para un incremento de 100rpm => T=14,28ms
     */
    AD1CON1 = 0x0000;    // Turn off ADC module
    AD1CON1 = 0x0004;    // 16-bit integer, manual trigger, Auto start sampling
    AD1CON2 = 0x6400;    // +Vref=AVdd/-Vref=AVref-, scan MUXA, interrupt at 1° sample/convert, 16 word buffer, only MUXA
    AD1CON3 = 0x0000;    // Tad from Fpb, Tad = 2*Fpb //Configure ADC conversion clock
    AD1CHS = 0x0000;     // Negative input CHA -Vref, resto no importa (scan)
    AD1CSSL = 0x0001;    // Scan inpunts AN0
    // configure 10-bitADC interrupt
    IPC6SET = 0x07000000;    // prioridad 1, subprioridad 3
    IFS1CLR = 0x00000002;    // bajo la bandera ADC

    // configure Input Capture
    /*
     *  Configurado para el uso con un sensor Hall 0.5 Hz/rpm
     */
    IC1CON = 0x0000;         // Apaga IC1
    IC1CON = 0xC001;         // Enciende IC1, FRZ, 16 bit, timer3, interrupción x captura, edge detect mode
    // configure IC1 interrupt
    IPC1SET = 0x00001800;    // prioridad 6, subprioridad 0
    IFS0CLR = 0x00000020;    // bajo la bandera IC1

    // configure timer 3
    /*
     * como timer para la cuenta de frecuencia del IC1
     */
    T3CON = 0;           // para timer 3 operation
    T3CONSET = 0x0000;   // TMR3 off, prescale 1:1, 16 bit mode, internal clock source
    PR3 = 0xFFFF;        // set period register to max, load PR3
    TMR3 = 0;            // Borra regsitro de cuenta

    // configure output compare 5
    /*
     * PWM para la velocidad del motor (frec max Portescap 275 Hz) y deteccion de
     * sobretemperatura
     */
    OC5CON = 0x0000;     // apaga OC1 module
    OC5R = 0x4705;       // Inicializa el reg 1 de comparacion
    OC5RS = 0x4705;      // Inicializa el reg 2 de para cambio de ciclo 50%
    OC5CON = 0x0007;     // Configura modo PWM, 16-bit, pin Fault input, TMR2
    // configure timer 2
    T2CON = 0x0000;      // para timer 2, 16 bit mode, prescale 1:1, internal clock source
    PR2 = 0x8E0A;        // Establece el periodo T=3.635e-3
    TMR2 = 0;            // Borra regsitro de cuenta
    // configura Timer 2 interrupt
    IPC2SET = 0x0000000C;    // prioridad 3, subriordad 0
    IFS0CLR = 0x00000100;    // bajo la bandera timer2
    // configure interrupt output comapre 5
    IPC5SET = 0x01000000;    // prioridad 4, subprioridad 0
    IFS0CLR = 0x00400000;    // Bajo la bandera de la int output compare

    // configure timer 4
    /*
     * Mide el n° pasos dados por el actuador dando una interrupcion cada 10 pasos
     * Portescap 1 step = 0.0254 mm, por la configuracion del drv8805 hay un flanco de
     * bajada cada cuatro pasos (0.1016 mm) cada 10 flancos se genera la interrupcion,
     * cada 1.016 mm
     */
    T4CON = 0;           // para timer 4 operation
    T4CONSET = 0x0002;   // TMR4 off, 16 bit mode, prescale 1:1, external clock source
    PR4 = 0x0009;        // set period register to 10 steps(1.016mm), load PR4
```

```
    TMR4 = 0;             // Borra regsitro de cuenta
    // configure timer 4 16-bit interrupt
    IPC4SET = 0x0000000D;   // prioridad 3, subprioridad 1
    IFS0CLR = 0x00010000;   // bajo la bandera

    // configure timer 5
    /*
     * Periodo de envio por busCAN, 1 seg-0x9896 // 14.5ms-0x0237
     * controla el tiempo de espera entre mensajes al crear una tabla nueva.
     */
    T5CON = 0;            // para timer 5 operation
    T5CONSET = 0x0070;    // TMR5 off,  bit mode, prescale 1:256, internal clock source
    PR5 = 0x0237;         // set period register, load PR5
    TMR5 = 0;             // Borra regsitro de cuenta TMR5
    // configure timer 5 16-bit interrupt
    IPC5SET = 0x0000000C;   // prioridad 3, subprioridad 0
    IFS0CLR = 0x00100000;   // bajo la bandera

    // configure CAN1 interrupt
    IPC11SET = 0x00040000;   // prioridad 1, subprioridad 0
    IFS1CLR = 0x04000000;    // limpia flag CAN1

    // configure external 2 interrupt
    /*
     * Pulsador del boton On/Off
     */
    INTCONCLR = 0x00000004;  // activa por flanco de bajada
    IPC2SET = 0x08000000;    // prioridad 2, subprioridad 0
    IFS0CLR = 0x00000800;    // bajo la bandera EXT2

    // configure external 3 interrupt
    /*
     * Para conectar el sensor fin de carrera
     */
    INTCONCLR = 0x00000008;  // activa por flanco de bajada
    IPC3SET = 0x08000000;    // prioridad 2, subprioridad 0
    IFS0CLR = 0x00008000;    // bajo la bandera EXT3

    // enable interrupts
    T2CONSET = 0x8000;       // Start timer 2
    T3CONSET = 0x8000;       // start timer 3
    T4CONSET = 0x8000;       // start timer 4
    T5CONSET = 0x8000;       // start timer 5
    IEC0SET = 0x00008800;    // enable EXT2,EXT3
    IEC0SET = 0x00400000;    // enable output compare 5
    IEC0SET = 0x00110100;    // enables T2, T4, T5 intrrupt
    IEC0SET = 0x00000020;    // enable IC1
    IEC1SET = 0x00000002;    // enables ADC interrupt
    IEC1SET = 0x04000000;    // habilita interrupcion CAN1

}
```

```
/*****************************************************************/
/*  Creación:                                                    */
/*    · Jaime García Padilla                                     */
/*****************************************************************/
/*  CAN_setup_8.c                                                */
/*  Fichero con las rutinas de inicialización                    */
/*  Creación:   octubre-2012                                     */
/*  Modificaciones:                                               */
/*  Tres FIFOs, control de errores, overflow y estado            */
/*****************************************************************/
#include "CAN.h"
#include "GenericTypeDefs.h"

#define SID1 0x001      /*SID*/
#define SID2 0x002      /*SID*/
#define SID3 0x003      /*SID*/
#define MAXTX 0x8       /*nº de octetos a enviar*/

// Estructura de la tabla
typedef union{
    struct{
        unsigned rpm:16;    /* rpm salto a siguiente posicion */
        unsigned mm:8;      /* distancia en mm de la posicon */
        unsigned maxposc:8; /* maximo nº de posiciones */
    };
    UINT32 messageWord;
}TABLA;
// Estructura del dato a enviar
typedef union{
    struct{
        unsigned rpm:16;    /* rpm actuales */
        unsigned TPH:8;     /* poscion en % del TPH actual*/
        unsigned mm:8;      /* distancia actual */
        unsigned n_mapa:4;  /* nº mapa en uso */
        unsigned :12;
        unsigned status:4;  /* si se introduce algun mensaje adicional */
        unsigned :4;
        unsigned estado:4;  /* estado actual del calculador */
        unsigned :4;
    };
    UINT messageWord[2];
}DATOTX;
// Estructura del dato recibido
typedef union{
    struct{
        unsigned :32;       /* vacio CAN */
        unsigned :32;       /* vacio CAN*/
        unsigned rpm:16;    /* rpm nuevas */
        unsigned mm:8;      /* distancia nueva*/
        unsigned maxposc:8; /* posiciones maximas o posicion a cambiar */
        unsigned :16;       /* vacio CAN */
        unsigned extra:8;   /* orden de recepcion */
        unsigned orden:8;   /* informacion adicional a la orden */
    };
    UINT32 messageWord[4];
}DATORX;
// variables globales
extern unsigned int CANFifoMensajeBuffer[120];   /*nº total del words en las FIFOs*/


void CAN_config(void)
{
    CAN_BIT_CONFIG canBitConfig;        /*para configurar el baudrate*/
    int n = 3;                          /*numero de FIFOs*/

    CANSetOperatingMode(CAN1,CAN_CONFIGURATION);            /*solicita modo config*/
    while(CANGetOperatingMode(CAN1) != CAN_CONFIGURATION);  /*espera entrar en config*/

    /*direccion base de las fifo del CAN0 y tamaño total*/
    CANAssignMemoryBuffer(CAN1, CANFifoMensajeBuffer, (n * 10 *  CAN_TX_RX_MESSAGE_SIZE_BYTES ));

    /*FIFO TX en CAN0, FIFO0, tamaño 10 mensajes, deshabilitada la respuesta remota, prioridad max*/
    CANConfigureChannelForTx(CAN1, CAN_CHANNEL0, 10, CAN_TX_RTR_DISABLED, CAN_HIGHEST_PRIORITY);

    /*FIFO RX en CAN0, FIFO1, tamaño 10 mensajes, recibe mensaje entero*/
    CANConfigureChannelForRx(CAN1, CAN_CHANNEL1, 10, CAN_RX_FULL_RECEIVE);

    /*FIFO RX en CAN0, FIFO2, tamaño 10 mensajes, recibe mensaje entero*/
    CANConfigureChannelForRx(CAN1, CAN_CHANNEL2, 10, CAN_RX_FULL_RECEIVE);

    /*configuracion del filtro*/
    CANConfigureFilter(CAN1, CAN_FILTER1,  SID1, CAN_SID);
    CANConfigureFilter(CAN1, CAN_FILTER2,  SID2, CAN_SID);

    /*configuracion de la mascara*/
    CANConfigureFilterMask(CAN1, CAN_FILTER_MASK1, 0x7FF, CAN_SID, CAN_FILTER_MASK_IDE_TYPE);

    /*union filtro, mascara y FIFO*/
    CANLinkFilterToChannel(CAN1, CAN_FILTER1, CAN_FILTER_MASK1, CAN_CHANNEL1);
    CANLinkFilterToChannel(CAN1, CAN_FILTER2, CAN_FILTER_MASK1, CAN_CHANNEL2);
    CANEnableFilter (CAN1, CAN_FILTER1, TRUE);
    CANEnableFilter (CAN1, CAN_FILTER2, TRUE);

    /*configuracion BaudRate*/
    /*
     * Hay que saber el Tq delay del sistema para repartir mejor los Tq ver datasheet
     * Sample poit al 70% (PS2=3TQ -> 30% del total Tbaud)
     * Ftq=N*Fbaud
     * BRP= Fsys/(2*Ftq)
     * Fbaud=125kbps; Fsys=80MHz; N=10
     */
    canBitConfig.phaseSeg1Tq        = CAN_BIT_3TQ;
    canBitConfig.phaseSeg2Tq        = CAN_BIT_3TQ;
    canBitConfig.propagationSegTq   = CAN_BIT_3TQ;
    canBitConfig.sample3Time        = FALSE;
```

```
    canBitConfig.syncJumpWidth     = CAN_BIT_1TQ;
    canBitConfig.phaseSeg2TimeSelect  = TRUE;

    CANSetSpeed(CAN1,&canBitConfig,80000000,125000);    /*80MHz, 125Kbts/s*/

    CANEnableModuleEvent(CAN1, ( CAN_RX_EVENT | CAN_SYSTEM_ERROR_EVENT
            | CAN_RX_OVERFLOW_EVENT | CAN_BUS_ERROR_EVENT), TRUE);       /* habilito el evento TX, RX y 3 eventos de error */
    CANEnableChannelEvent(CAN1, CAN_CHANNEL1, (CAN_RX_CHANNEL_NOT_EMPTY | CAN_RX_CHANNEL_OVERFLOW), TRUE);      /* habilito el evento de FIFO 1 no vacia */
    CANEnableChannelEvent(CAN1, CAN_CHANNEL2, (CAN_RX_CHANNEL_NOT_EMPTY | CAN_RX_CHANNEL_OVERFLOW), TRUE);      /* habilito el evento de FIFO 1 no vacia */
    CANSetOperatingMode(CAN1,CAN_NORMAL_OPERATION);              /*solicita modo normal*/
    while(CANGetOperatingMode(CAN1) != CAN_NORMAL_OPERATION);   /*espera entrar en normal*/

    CANClearModuleEvent(CAN1, CAN_OPERATION_MODE_CHANGE_EVENT);   /* Borramos flag */
}
/*
 * BOOL CAN_transmit(DATOTX *dato)
 *
 * funcion que transmite un dato con una configuracion predeterminada
 * Devuelve:
 * TRUE : si la transmision se produjo sin error en canal
 * FALSE : si se produjo algun error en canal
 */
BOOL CAN_transmit(DATOTX *dato)
{
    CANTxMessageBuffer *buffer;     /* direcion de la FIFO en uso */
    int SID = SID3;
    BOOL IDE = FALSE;
    int EID;
    char DLC = MAXTX;
    CAN_TX_CHANNEL_CONDITION status;

    // Prepra y envia un dato
    buffer = CANGetTxMessageBuffer(CAN1, CAN_CHANNEL0);  /* leo direccion */
    if (buffer != NULL){    /* carga mensaje si no esta llena la FIFO */
        buffer->msgSID.SID = SID;               /* identificador SID */
        buffer->msgEID.IDE = IDE;               /* tipo mensaje SID */
        if(IDE == TRUE){    /*si el mensaje es EID*/
            buffer->msgEID.EID = EID;           /* identificador EID */
        }
        buffer->msgEID.DLC = DLC;               /* nº bytes de datos */
        buffer->messageWord[2] = dato->messageWord[0];       /* bytes de datos */
        if (DLC > 0x04){    /*si hay mas de 4 bytes*/
            buffer->messageWord[3] = dato->messageWord[1];   /* bytes de datos */
        }
        //CANClearModuleEvent(CAN1, CAN_INVALID_RX_MESSAGE_EVENT);    /* limpia flag invalid Rx */
        CANUpdateChannel(CAN1, CAN_CHANNEL0);    /* actualiza punteros internos */
        CANFlushTxChannel(CAN1, CAN_CHANNEL0);   /* envia el mensaje */
        while( CANGetTxChannelCondition(CAN1, CAN_CHANNEL0) == CAN_TX_CHANNEL_TRANSMITTING);      /* Espera a que termine de transmitir */
    }

    // Comprueba el estado de la FIFO
    status = CANGetTxChannelCondition(CAN1, CAN_CHANNEL0);
    if((status & (CAN_TX_CHANNEL_ARBITRATION_LOST | CAN_TX_CHANNEL_ERROR)) != 0){
        if (status == CAN_TX_CHANNEL_ERROR ){
            // hubo error en la transmision
            CANFlushTxChannel(CAN1, CAN_CHANNEL0);   /* reenvia el mensaje */
            while( CANGetTxChannelCondition(CAN1, CAN_CHANNEL0) == CAN_TX_CHANNEL_TRANSMITTING);      /*Espera a que termine de transmitir*/
            if(CANGetTxChannelCondition(CAN1, CAN_CHANNEL0) == CAN_TX_CHANNEL_ERROR ){
                return FALSE;
            }
            else{
                return TRUE;
            }
        }
        else{
            // se ha "perdido" la FIFO o error en Tx, se debe reiniciar la FIFO
            CANResetChannel(CAN1,CAN_CHANNEL0);
            while(CANIsChannelReset(CAN1,CAN_CHANNEL0) != TRUE);     /* espera a reset FIFO 0 */
            return FALSE;   /* error en la transmision*/
        }
    }
    else{
        return TRUE;    /* transmision correcta */
    }
    return FALSE;   /* error en la transmision*/
}
/*
 * CAN_EVENT_CODE CAN_recive(DATORX *dato, CAN_MODULE_EVENT modEvent)
 *
 * funcion que recive un dato del bus CAN y retorna el numero del filtro golpeado
 */
CAN_EVENT_CODE CAN_recive(DATORX *dato, CAN_MODULE_EVENT modEvent)
{
    CANRxMessageBuffer * rxMsg;          /* direcion de la FIFO en uso */
    CAN_CHANNEL_EVENT  ChannelRxEvent;   /* identificador de eventos */
    CAN_EVENT_CODE eventCode;            /* identificado de la FIFO*/

    if ((modEvent&CAN_INVALID_RX_MESSAGE_EVENT) == 0){
        // mensaje sin errores
        eventCode = CANGetPendingEventCode(CAN1);    /* identifico la FIFO del evento */

        ChannelRxEvent = CANGetChannelEvent(CAN1,eventCode);   /* recoge evento en FIFO 1 */
        /*Compruebo que hay un mensaje para leer*/
        if ( ChannelRxEvent == CAN_RX_CHANNEL_NOT_EMPTY ){   /* si la FIFO no esta vacia */
            /* Obtengo la dirección del buffer de lectura */
            rxMsg = (CANRxMessageBuffer *)CANGetRxMessage(CAN1, eventCode);
            dato->messageWord[0] = rxMsg -> messageWord[0];
            dato->messageWord[1] = rxMsg -> messageWord[1];
            dato->messageWord[2] = rxMsg -> messageWord[2];
            dato->messageWord[3] = rxMsg -> messageWord[3];
            /* Actualizo el puntero al buffer de mensajes */
            CANUpdateChannel(CAN1, eventCode);    /* recepcion correcta */
            CANClearModuleEvent(CAN1, CAN_RX_EVENT);     /*limpia flag de recepción*/

            return eventCode;    /* recepcion correcta */
```

```
        }
    }
    else{
        // invalid message recibido
        CANClearModuleEvent(CAN1, CAN_INVALID_RX_MESSAGE_EVENT);      /* limpia flag invalid Rx*/
    }
    return eventCode;   /* recepcion incorrecta */
}
/*
 * void SystemErrorEvent (void)
 *
 * funcion que gestiona los errores de sistema CAN1 actuando para corregirlos
 * comunicando por PORTE el codigo del error
 */
void SystemErrorEvent (void){
    CAN_EVENT_CODE eventCode;   /* codigo de error*/

    PORTESET = 0x08;                            /* codigo para el error */
    eventCode = CANGetPendingEventCode(CAN1);   /* guardo evento pendiente */
    CANEnableModule(CAN1, FALSE);               /* deshabilito el modulo CAN */
    while(CANIsActive(CAN1) == TRUE);           /* espera descativacion del modulo */
    CANClearModuleEvent(CAN1, CAN_SYSTEM_ERROR_EVENT);   /* Borramos flag */
    // identifico la fuente del error
    switch(eventCode){
        case CAN_BUS_BANDWIDTH_ERROR:
            // otro nodo envia mas rapido de lo que proceso lo recibido
            // no se arranca de nuevo el modulo CAN
            break;
        case CAN_ADDRESS_ERROR_EVENT:
            // invalid base adress or message destination ilegal
            // se arranca de nuevo el modulo CAN
            CANEnableModule(CAN1, TRUE);     /*habilito el modulo CAN*/
            CAN_config();                    /*configura el CAN*/
            break;
        default:
            // invalid message recived
            break;
        }
}
/*
 * void ErrorBusEvent (void)
 *
 * funcion que gestiona los errores del bus CAN1 indicando un codigo por el
 * PORTE para indicar el estado del bus.
 */
void ErrorBusEvent (void){
    CAN_ERROR_STATE errorState;

    CANClearModuleEvent(CAN1, CAN_BUS_ERROR_EVENT);  /* ¿necesito borrar el flag? */
    errorState = CANGetErrorState(CAN1);

    switch (errorState){
        case CAN_RX_WARNING_STATE:
            // activa led
            PORTESET = 0x01;        /*codigo para el estado*/
            break;
        case CAN_TX_WARNING_STATE:
            // activa led
            PORTESET = 0x01;        /*codigo para el estado*/
            break;
        case CAN_RX_BUS_PASSIVE_STATE:
            // activa led
            PORTESET = 0x02;        /*codigo para el estado*/
            break;
        case CAN_TX_BUS_PASSIVE_STATE:
            // activa led
            PORTESET = 0x02;        /*codigo para el estado*/
            break;
        case CAN_TX_BUS_OFF_STATE:
            // activa led
            PORTESET = 0x04;        /*codigo para el estado*/
            break;
        default:

            break;
    }
}
/*
 * BOOL RxOverflowEvent (void)
 *
 * funcion que gestiona overflow de FIFO RX, bajando la bandera OVF, reiniciando
 * el canal y retornando TRUE si se produjo overflow
 */
BOOL RxOverflowEvent (void){
    CAN_CHANNEL_MASK channelOverflowEvent;

    channelOverflowEvent = CANGetAllChannelOverflowStatus(CAN1);      /* status de overflows */

    if((channelOverflowEvent & CAN_CHANNEL1_MASK) != 0){      /* identifica la FIFO1 */
        // overflow Channel 1 de CAN1
        CANClearChannelEvent(CAN2, CAN_CHANNEL1, CAN_RX_CHANNEL_OVERFLOW);    /* limpia la bandera OVF */
        CANResetChannel(CAN1,CAN_CHANNEL1);                                   /* reset CHANNEL1 */
        while(CANIsChannelReset(CAN1,CAN_CHANNEL1) != TRUE);                  /* espera reset de la fifo */
        return TRUE;
    }
    else {      /* solo puede ser la FIFO2 */
        // overflow Channel 2 de CAN1
        CANClearChannelEvent(CAN2, CAN_CHANNEL2, CAN_RX_CHANNEL_OVERFLOW);    /* limpia la bandera OVF */
        CANResetChannel(CAN1,CAN_CHANNEL2);                                   /* reset CHANNEL2 */
        while(CANIsChannelReset(CAN1,CAN_CHANNEL2) != TRUE);                  /* espera reset de la fifo */
        return TRUE;
    }
    return FALSE;
}
```

```
/*********************************************************************/
/*  Creación:                                                        */
/*    · Jaime García Padilla                                         */
/*********************************************************************/
/*  PrigMem_7.c                                                      */
/*  Fichero con las rutinas de inicialización                        */
/*  Creación:   enero-2013                                           */
/*  Modificaciones:                                                  */
/*  se han mejorado las estructuras de datos, hay funciones para la  */
/*  actualizacion y lecura de la tabla en memoria                    */
/*********************************************************************/
#include <stdlib.h>
#include "p32xxxx.h"
#include "kmem.h"
#include "GenericTypeDefs.h"

#define MAXMAPAS  6          /* max numero de mapas motor*/
#define MAXTABLA  14         /* nº maximo de posiciones en tabla */
#define MINESCALON 500       /* incremento minimo de rpm por escalon */

// Estructura de la tabla
typedef union{
    struct{
        unsigned rpm:16;     /* rpm salto a siguiente posicion */
        unsigned mm:8;       /* distancia en mm de la posicon */
        unsigned maxposc:8; /* maximo nº de posiciones */
    };
    UINT32 messageWord;
}TABLA;
// Estructura del dato a enviar
typedef union{
    struct{
        unsigned rpm:16;     /* rpm actuales */
        unsigned TPH:8;      /* poscion en % del TPH actual*/
        unsigned mm:8;       /* distancia actual */
        unsigned n_mapa:4;   /* nº mapa en uso */
        unsigned :12;
        unsigned status:4;   /* si se introduce algun mensaje adicional */
        unsigned :4;
        unsigned estado:4;   /* estado actual del calculador */
        unsigned :4;
    };
    UINT messageWord[2];
}DATOTX;
// Estructura del dato recibido
typedef union{
    struct{
        unsigned :32;        /* vacio CAN */
        unsigned :32;        /* vacio CAN*/
        unsigned rpm:16;     /* rpm nuevas */
        unsigned mm:8;       /* distancia nueva*/
        unsigned maxposc:8; /* posiciones maximas o posicion a cambiar */
        unsigned :16;        /* vacio CAN */
        unsigned extra:8;    /* orden de recepcion */
        unsigned orden:8;    /* informacion adicional a la orden */
    };
    UINT32 messageWord[4];
}DATORX;
// variables globales
extern int MAXPOSC;                  /* maximo numero de posiciones */
extern const UINT __attribute__((space(prog))) ADDRESS_TABLA0;   /* direccion de almacenamiento */
extern DATORX CANBufferRx;           /*buffer de recepcion del modulo CAN*/
extern DATOTX CANBufferTx;           /*buffer de escritura del modulo CAN*/
// prototipos de funciones
int CreaTabla (TABLA *tabla);                        /* Crea la tabla de posiciones */
UINT GuardaTabla(TABLA *tabla, const UINT memWrite); /* guarda tabla en Flash */
UINT CargaTabla(TABLA *tabla, const UINT memRead);   /* carga tabla de la Flash */
BOOL ModificaTabla (TABLA *tabla);                   /* modifica una posicion de la tabla */
UINT CargaTPH(UINT *TPH, const UINT memRead);        /* carga TPH de la Flash */
UINT GuardaTPH( const UINT memWrite);                /* guarda TPH en la Flash */
/*
 * int CreaTabla (TABLA *tabla)
 *
 * Funcion que permite ir creando una tabla nueva y almacenarla en memoria no volatil
 * devuelve 1 si la operacion se realizo correctamente, en caso de estar esperando
 * un nuevo dato devuelve 0, en caso de 3 errores devuelve 2 y carga anterior tabla que
 * estuviera en memoria no volatil y devuelve 3 en caso de dato fuera de rango
 * La tabla esta configurada, con incrementos minimos de 100rpm por escalon
 */
int CreaTabla (TABLA *tabla){
    static BOOL nueva = TRUE;
    static int i, error=0;
    static int n_mapa, dir;

    if (nueva){
        if (CANBufferRx.extra<MAXMAPAS ){   /* nº mapa existente */
            // dentro del nuemero de mapas maximo
            if ((1 <= CANBufferRx.maxposc-1)&&(CANBufferRx.maxposc <= MAXTABLA)){
                /* hasta MAXTABLA posic max y 2 posic min*/
                MAXPOSC = CANBufferRx.maxposc;  /* nuevas max posiciones */
                tabla[0].mm = 0;                /* posicion inicial */
                tabla[0].rpm = 8000;            /* rpm inicial */
                tabla[0].maxposc = MAXPOSC;     /* maximo de posiciones */
                n_mapa = CANBufferRx.extra;     /* direccion Flash */
                nueva = FALSE;
                error = 0;
                i = 1;
            }

        }
```

```
        ,
        else{
            i = MAXPOSC + 1;      /* evita el siguiente caso */
            error++;
        }
    }
    else if (i < MAXPOSC){       /* rellena la tabla */
        if (((1 <= CANBufferRx.mm)&&(CANBufferRx.mm <= 180))&&
                ((8000 <= CANBufferRx.rpm)&&(CANBufferRx.rpm <= 15000))){
            /* valores dentro de rango */
            if ((CANBufferRx.rpm >= tabla[i-1].rpm+MINESCALON)&&(CANBufferRx.mm > tabla[i-1].mm)){
                /* los datos son correctos para conformar tabla*/
                tabla[i].rpm = CANBufferRx.rpm; /* guarda datos */
                tabla[i].mm = CANBufferRx.mm;
                tabla[i].maxposc = MAXPOSC;
                error = 0;
                i++;          /* siguiente posicion */
            }
            else
                error++;
        }
        else
            error++;
        if (i == MAXPOSC){   /* tabla completa */
            nueva = TRUE;                 /* prepara para nueva tabla */
            error = 0;                    /* reinicia cuenta errores */
            dir = ADDRESS_TABLA0 +(n_mapa*0x1000);
            if(GuardaTabla(tabla, dir)==0){  /* guarda en memoria no volatil */
                CANBufferTx.n_mapa = n_mapa;     /* actualiza n_mapa actual */
                /* notifico por CANTX tabla OK */
                return 1;    /* tabla nueva guardada */
            }
        }
    }
    if (error >= 3){     /* error al recibir datos */
        nueva = TRUE;                     /* prepara para nueva tabla */
        error = 0;                        /* reinicia cuenta errores */
        dir = ADDRESS_TABLA0 +(CANBufferTx.n_mapa*0x1000);    /* direccion de mapa motor en uso */
        CargaTabla(tabla, dir);           /* recupera tabla de la Flash*/
        /* notifico por CANTX */
        return 2;    /* carga tabla anterior */
    }
    if (error != 0)
        return 3;    /* dato fuera de rango */
    return 0;    /* dato ok */
}
/*
 * BOOL ModificaTabla (TABLA *tabla)
 *
 * Funcion que permite modificar una posicon de la tabla y almacenarla en memoria
 * no volatil, devuelve 1 si la operacion se realizo correctamente, en caso de
 * error devuelve 0 y no modifica lo que estuviera en memoria no volatil
 */
BOOL ModificaTabla (TABLA *tabla){
    int dir;

    if ((2 <= CANBufferRx.maxposc)&&(CANBufferRx.maxposc <= MAXPOSC)){   /* posicion a cambiar dentro de la tabla */
        if ((CANBufferRx.rpm >= tabla[CANBufferRx.maxposc-2].rpm+MINESCALON)&&
                (CANBufferRx.rpm <= tabla[CANBufferRx.maxposc].rpm-MINESCALON)){
            // nuevas rpm de la poscion dentro del rango de rpm que le queda
            if ((CANBufferRx.mm > tabla[CANBufferRx.maxposc-2].mm)&&(CANBufferRx.mm < tabla[CANBufferRx.maxposc].mm)){
                // nuevos mm de la poscion dentro del rango de mm que le queda
                tabla[CANBufferRx.maxposc-1].rpm = CANBufferRx.rpm; /* guarda datos */
                tabla[CANBufferRx.maxposc-1].mm = CANBufferRx.mm;
                tabla[CANBufferRx.maxposc-1].maxposc = MAXPOSC;
                dir = ADDRESS_TABLA0+(CANBufferTx.n_mapa*0x1000);    /* direccion de mapa motor en uso */
                GuardaTabla(tabla, dir);   /* guarda en memoria no volatil */
                /* notifico por CANTX tabla OK */
                return 1;
            }
        }
    }
    return 0;
}

/*
 * UINT NVMUnlock (UINT nvmop)
 *
 * Funcion Unlock de Flash memory
 * Permite el almacenamiento en memoria no volatil, para interrupciones y mete
 * la secuencia de llaves.
 */
UINT NVMUnlock (UINT nvmop){

    IEC0CLR = 0xFFFFFFFF;    /* desactiva las interrupciones */
    IEC1CLR = 0xFFFFFFFF;    /* desactiva las interrupciones */
    NVMCON = nvmop;          /* enable Flash W/E operations and select flash operation to perform */
    NVMKEY = 0xAA996655;     /* write key */
    NVMKEY = 0x556699AA;     /* write key */
    NVMCONSET = 0x8000;      /* start the operation using the set register */
    while (NVMCON & 0x8000); /* wait for operation to complete */
    NVMCONCLR = 0x4000;      /* clear enable NVMWREN */
    IEC0SET = 0x00408820;    /* enable EXT2, EXT3, OC5, IC1 */
    IEC0SET = 0x00110100;    /* enables T2, T4, T5 intrrupt */
    IEC1SET = 0x04000002;    /* habilita interrupcion CAN1, enables ADC interrupt */

    return (NVMCON & 0x3000); /* return error status bits */
}
/*
 * UINT NVMWirteWord (void* address, UINT data)
```

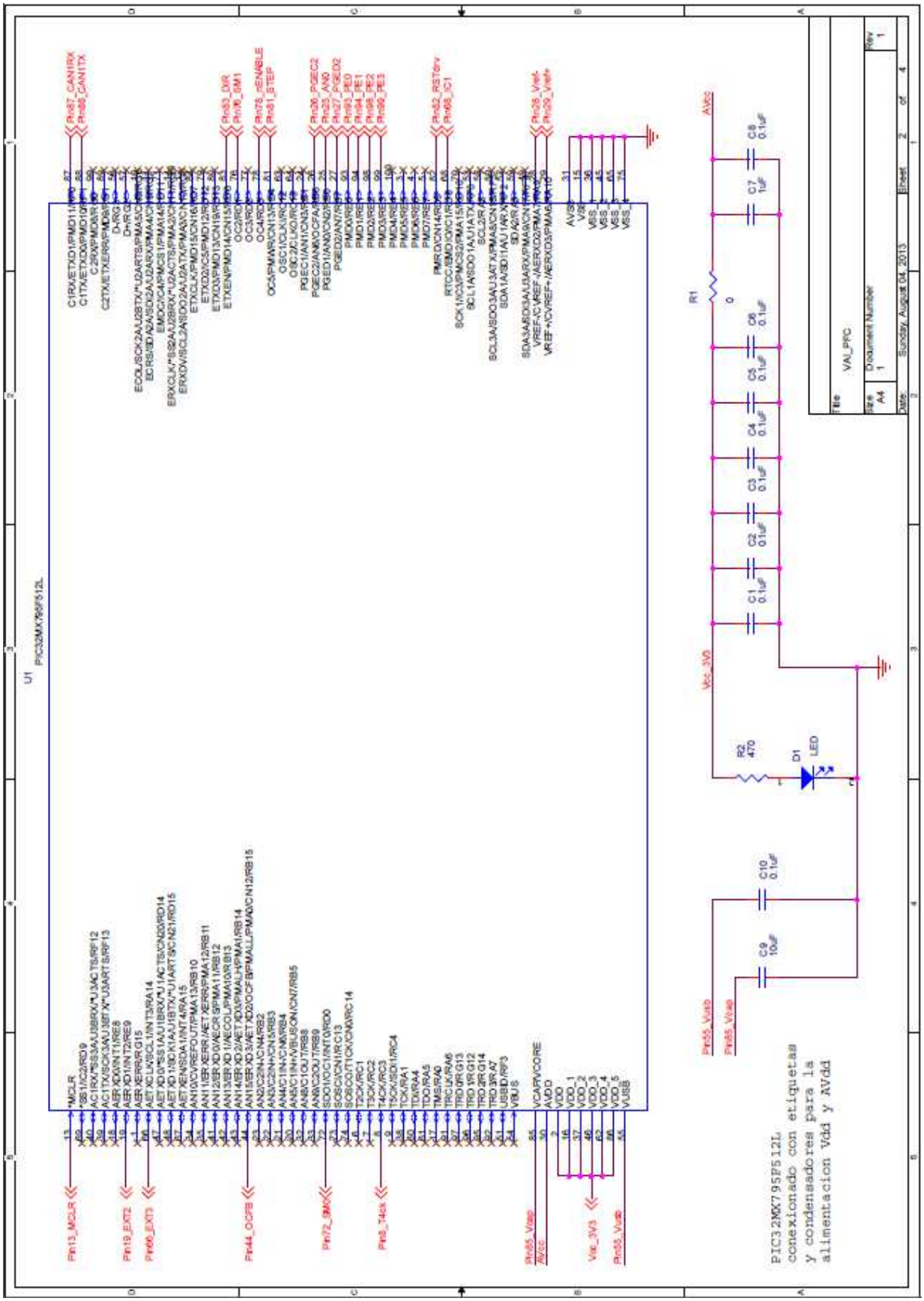
```
 *
 * Funcion programacion de un Word en Flash memory
 * Almacena un word en memoria no volatil
 */
UINT NVMWirteWord (void* address, UINT data){
    UINT res;

    NVMDATA = data;                 /* load data into NVMDATA register */
    NVMADDR = KVA_TO_PA(address); /* load address to program into NVMADDR register */
    res = NVMUnlock (0x4001);       /* Unlock and write word */

    return res;          /* return result */
}
/*
 * UINT NVMErasePage(void* address)
 *
 * Funcion que borra una pagina de Flash memory
 * Borra una pagina de memoria no volatil (minimo de memoria que se puede borrar)
 */
UINT NVMErasePage(void* address){
    UINT res;

    NVMADDR = KVA_TO_PA(address); /* set NVMADDR to the strat address of page to erase */
    res = NVMUnlock (0x4004);       /* Unlock and erase page */

    return res;          /* return result */
}
/*
 * UINT GuardaTabla(TABLA *tabla, const UINT memWrite)
 *
 * Funcion guarda una tabla en direccion de memoria no volatil
 * Recorre una tabla almacenandola en memoria no volatil
 */
UINT GuardaTabla(TABLA *tabla, const UINT memWrite){
    UINT res, data, dir;
    int i = 0;

    res = NVMErasePage(( void*)memWrite);
    while((i < MAXPOSC)&&(res == 0)){    /* posicion correcta y sin errores */
        /* escribe en Flash uno a uno los datos de la tabla */
        data = tabla[i].messageWord;     /* dato a guardar */
        dir = memWrite+(i*0x010);        /* direccion */
        res = NVMWriteWord(dir, data);   /* guarda el dato */
        i++;
    }
    return res; /* error */
}
/*
 * UINT CargaTabla(TABLA *tabla, const UINT memRead)
 *
 * Carga la tabla a partir de la posicion de memoria
 * Lee y recupera de memora no volatil una tabla guardada
 */
UINT CargaTabla(TABLA *tabla, const UINT memRead){
    UINT i = 0;
    UINT *ptrRead;

    ptrRead = (UINT*)memRead;   /* inicia puntero de lectura */
    do{
        tabla[i].messageWord = *(ptrRead+(i*0x04));
        i++;
    }while(i < tabla[0].maxposc);   /* carga la tabla */
    MAXPOSC = tabla[0].maxposc;     /* posiciones maximas */
    return 0;
}
/*
 * UINT GuardaTPH(UINT TPH, const UINT memWrite)
 *
 * Guarda en memoria no volatil el TPHMIN
 */
UINT GuardaTPH( const UINT memWrite){
    UINT dir;
    int res;

    if ((CANBufferRx.extra<=100)&&(CANBufferRx.extra>0)){    /* si el valor de tph esta en rango*/
        res = NVMErasePage(( void*)memWrite);
        dir = memWrite+0x010;                /* direccion */
        NVMWriteWord(dir, CANBufferRx.extra);  /* guarda el dato */
        return 1;           /* tphmin guardado*/

    }

}
/*
 * UINT CargaTPH(UINT *TPH, const UINT memRead)
 *
 * Carga la tphmin a partir de la posicion de memoria
 */
UINT CargaTPH(UINT *TPH, const UINT memRead){
    UINT *ptrRead;

    ptrRead = (UINT*)memRead;   /* inicia puntero de lectura */
    *TPH = *(ptrRead+0x04);      /* lee dato */
    return 0;
}
```

```
/*************************************************************/
/*  Creación:                                                */
/*    · Jaime García Padilla                                 */
/*************************************************************/
/*  drv8805_3.c                                              */
/*  Fichero con las rutinas de inicialización                */
/*  Creación:   octubre-2012                                 */
/*  Modificaciones:                                          */
/*  Gestiona el motor para poder manejarlo con 6 ordenes que lleva al  */
/*  a 6 estados diferentes para su funcionamiento            */
/*************************************************************/

#include "p32xxxx.h"
#include <plib.h>

// Estados del automata
#define nFAULT  0   /* estdo overtemp */
#define RESET   1   /* estado reset */
#define HOLD    2   /* mantiene poicion */
#define DIR1    3   /* direccion creciente */
#define DIR0    4   /* direccion decrecinte */
#define WAIT    5   /* estado de espera */

extern int Indice;


/*
 * BOOL ControlMotor(int estado)
 *
 * Funcion que gestiona el PORTD para el control del drv8805 devuelve TRUE si
 * drv8805 ha tenido algun error
 * Para bloquear el motor -> detener OC5
 * Para detener el motor -> PORTDSET = 0x08;// disable output drv8805
 */
BOOL ControlMotor(int estado){
    if (PORTDbits.RD5 == TRUE){ /* se reiniciaron las cuentas */
        PORTDCLR = 0x20;      /* libera reset drv8805 */
    }
    if (OC5CONbits.OCFLT == 0){
        switch (estado) {
            case nFAULT:    /* si ya no hay nFAULT */
                OC5CON = 0x0007;     /* Configura modo PWM, 16-bit, pin Fault input, TMR2 */
                PORTDCLR = 0x03;     /* modo 0 de operadion drv8805 */
                PORTDCLR = 0x08;     /* enable output drv8805 */
                return TRUE;
                break;
            case RESET:
                Indice = 0;          /* inicio del perfil */
                PORTDCLR = 0x03;     /* modo 0 de operadion drv8805 */
                PORTDCLR = 0x40;     /* direccion decreciente */
                PORTDCLR = 0x08;     /* enable output drv8805 */
                OC5CONSET = 0x8000;  /* Enable OC1 */
                break;
            case HOLD:
                Indice = 0;          /* inicio del perfil */
                PORTDSET = 0x08;     /* Disable output drv8805 */
                break;
            case WAIT:
                // solo mientras espera recepcion de nueva tabla
                PORTDSET = 0x08;     /* Disable output drv8805 */
                break;
            case DIR1:
                Indice = 0;          /* inicio del perfil */
                PORTDSET = 0x40;     /* direccion creciente */
                PORTDCLR = 0x08;     /* enable output drv8805 */
                OC5CONSET = 0x8000;  /* enable OC5 */
                break;
            case DIR0:
                Indice = 0;          /* inicio del perfil */
                PORTDCLR = 0x40;     /* direccion decreciente */
                PORTDCLR = 0x08;     /* enable output drv8805 */
                OC5CONSET = 0x8000;  /* enable OC5 */
                break;
        }
    }
    return FALSE;
}
```

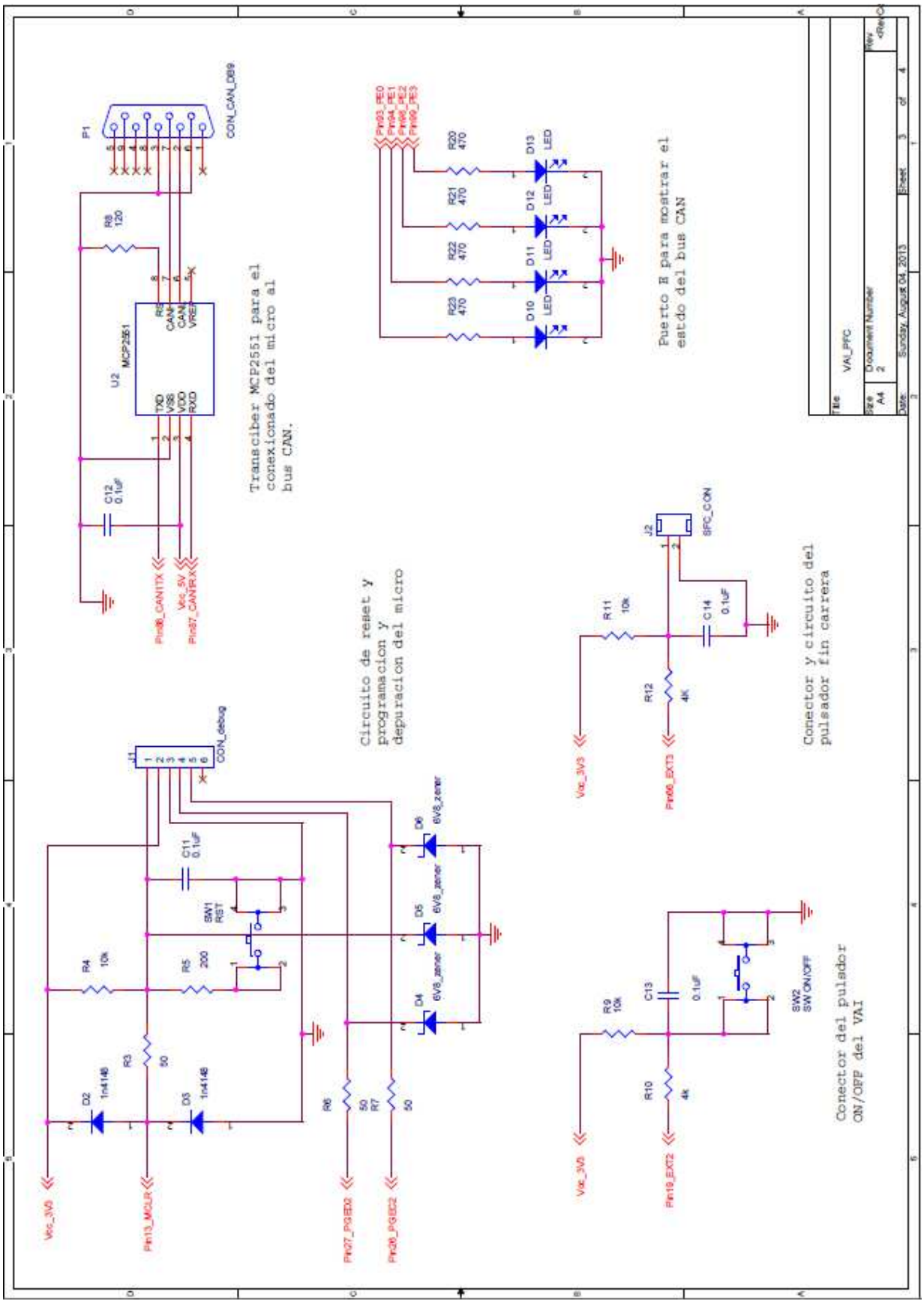
9. ANEXO III.

ESQUEMÁTICOS Y FOTOLITOS VAI4.

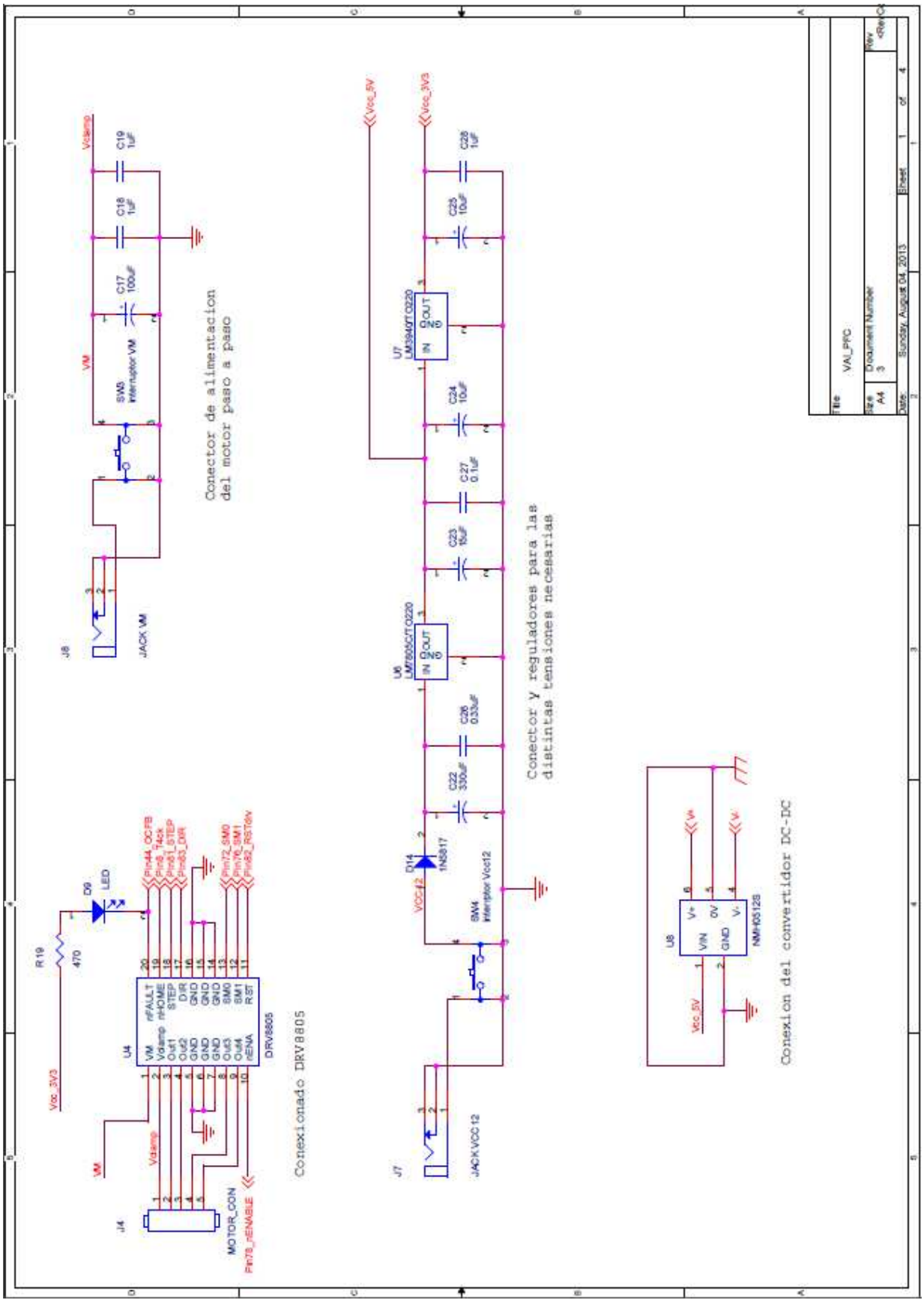
Esquemático de Capture, página 1.



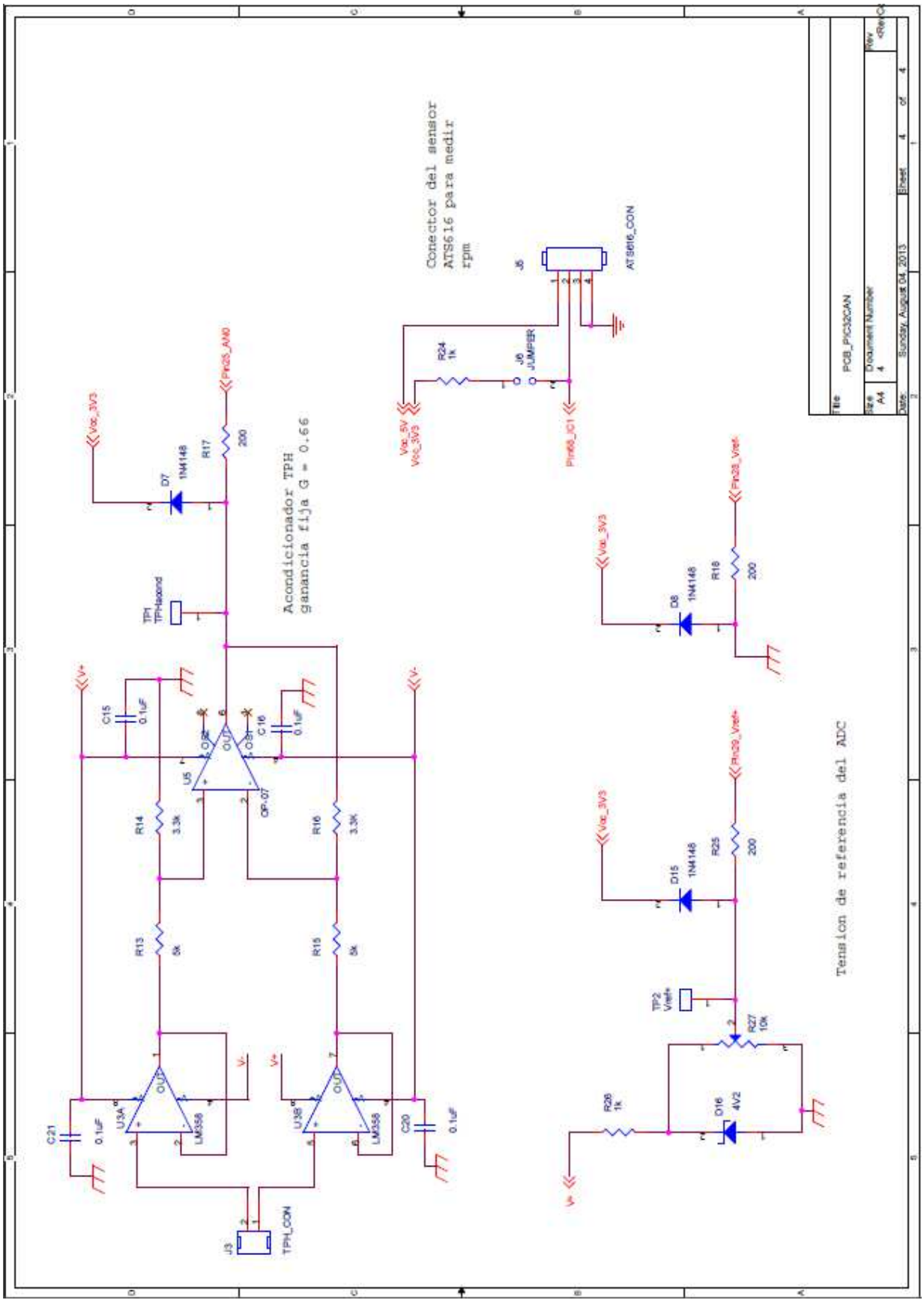
Esquemático de Capture, página 2.



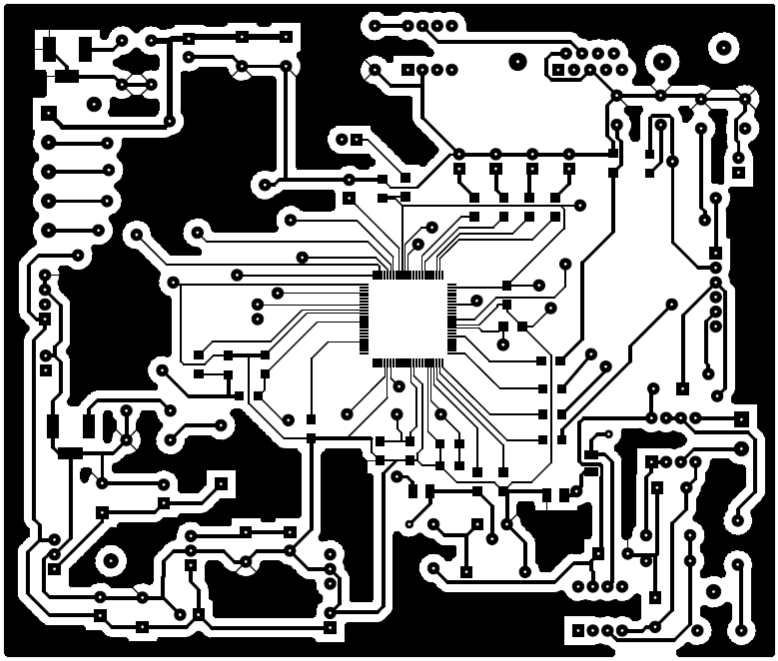
Esquemático de Capture, página 3.



Esquemático de Capture, página 4.



Layout, cara Top.



Layout, cara Bottom.